Template-Based Synthesis of Plan Execution Monitors

Thomas Reinbacher, and César Guzmán Alvarez

[1] Embedded Computing Systems Group, Vienna University of Technology, Austria
treinbacher@ecs.tuwien.ac.at
[2] Universidad Politecnica de Valencia, Spain
cguzman@dsic.upv.es

Abstract. In the robotics domain, the state of the world may change in unexpected ways during execution of a task. From a planning perspective, these discrepancies may render the currently executed plan invalid and thus need to be detected as soon as possible. We tackle this problem by translating the problem of plan execution monitoring to a runtime verification problem. We propose a template based framework that allows detecting changes of the state during both plan generation and plan execution. We integrated our approach into a domain-independent platform for planning, executing, and monitoring.

1 Introduction and Related Works

During plan execution, discrepancies between the expected and the actual state of the world can stem from various sources. Rigorous execution monitoring is thus required to i) detect errors, ii) communicate with a replanning module or iii) to collect training instances for a learning module.

Plan execution monitoring needs to deal with the problem of monitoring actions in terms of its preconditions and effects, which typically represents an abstraction of the real problem. Several systems [10, 12, 3] monitor the continued validity of a plan during execution using annotations (variables to be monitored) of the plan. However, these annotations are generated without considering the goals of the planning problem. The work of [11] improved this process by including the goals of the planning problem through a regression algorithm. We build on these ideas from annotation-based approaches but separate the annotations from the plan to use them to instantiate our generic templates. Systems described by [9, 14] perform plan execution monitoring only prior to the execution of an action but not during it's execution.

A technique with similar aims as plan execution monitoring is runtime verification [5] which has been successfully applied to a number of high-level programming languages. The authors of [15] report on an execution-monitoring approach with temporal logic. However, their specifications are defined as domain-specific formulas (Unmanned Aerial Vehicles), while we focus on a generic, domain-independent solutions, such as Mars Rovers, Blockworlds, Health, Fire-extinction, etc. Furthermore, their approach specifications are given manually, while we aim at offering

a great degree of automation by providing templates. While they build on a domain description language based on TAL (Temporal Action Logic), we build on the standardized language PDDL.

In this paper, we present a first approach to explore the benefits of using techniques from the runtime verification community to alleviate the problems that arise in the field of plan monitoring at it's execution time. While these two fields are typically seen as disconnected from each other, runtime verification provides a formal methods based backend to automatically generate executable monitors for a given plan. We suggest that this link is worth exploring and therefore advocate the idea of deploying runtime verification as a technique for plan execution monitoring. We sidestep the cumbersome task of compiling a specification for the monitoring goals by providing a set of generic templates, which allow to express temporal properties involving timing requirements. These templates are instantiated and configured through a set of parameters we automatically derive from the plan. Finally, we use results from the runtime verification community to automatically derive an executable observer. We show how this technique seamlessly integrates into the traditional planning approach, by implementing the technique into a multi-agent domain-independent planning, execution and monitoring architecture. In this paper we do not explain the main components of this architecture we only focus on the monitoring component.

2 Motivating Example: Mars Rovers

In this section, we elaborate on a possible Mars Rovers scenario[3] to detail the problem of monitoring and executing autonomous plans for a dynamic multi-agent system. In this system, two agents (i.e., rovers) are working in close proximity and need to cooperate in order to accomplish a mission.

The scenario assumes that there are two Mars Rovers ($\mathsf{R_a}$ and $\mathsf{R_b}$) and three waypoints $\{\mathsf{W_1}, \mathsf{W_2}, \mathsf{W_3}\}$, where $\mathsf{W_2}$ is the waypoint of the Lander L and the initial state of both rovers. Rovers $\mathsf{R_a}$ and $\mathsf{R_b}$ are working near each other, both equipped with a set of navigation cameras. $\mathsf{R_a}$ uses a microscopic camera to analyze rocks in waypoint $\mathsf{W_1}$ and sends the results to the lander, whereas $\mathsf{R_b}$ takes panoramic pictures of the surrounding terrain and communicates it to the lander in the waypoint $\mathsf{W_2}$. Communication with the lander and the satellite can be initiated from any waypoint $\mathsf{W_x}$. The rovers are constrained by limited energy, on-board data storage, downlink opportunities, as well as available bandwidth and time to complete observations.

These limitations complemented with the unpredictability of the environment, may cause unforeseen problems during the mission. In the following, we will show how our template based plan execution monitoring helps to detect such failures as early as possible.

The initial plan for the mission is provided from a planner agent, i.e., the control center on Earth. In a subsequent step, each plan is forwarded to a dedicated

[3] We work with the rover temporal domain as defined in the International Planning Competition (IPC).

decision support module, located in the planner agent, to derive the variables to be monitored during plan execution. The goal is to determine the information that needs to be monitored to guarantee a successful plan execution. We use an anytime regression approach to *automatically* select the variables to be observed by the monitoring module during the plan monitoring. We compute the variables to be monitored through an extension of the goal regression method proposed in [11]. In the following, we refer to this information as *monitor-parameters*.

Then, this plan is divided into a set of actions, which are, together with their respective goals, and their *monitor-parameters* sent to both rovers $\mathsf{R_a}$ and $\mathsf{R_b}$ to be executed. Each of the rovers executes its assigned plan which consists of sequentially executed actions. Let's suppose the first action planned to be executed by $\mathsf{R_a}$ is the action Navigate. The action navigates the rover from waypoint $\mathsf{W_2}$ to waypoint $\mathsf{W_1}$ within a scheduled duration of 10 time units.

The following are the *high-level* conditions and effects for the action Navigate, as dictated by the domain:

$$\text{preconditions} := \begin{cases} \text{at}(\mathsf{Ra}, W_2) \wedge \\ \text{energy}(\mathsf{Ra}) \geq 8 \wedge \\ \text{available}(\mathsf{Ra}). \end{cases}$$

$$\text{invariants} := \begin{cases} \text{can\_traverse}(\mathsf{Ra}, W_2, W_1) \wedge \\ \text{visible}(W_2, W_1). \end{cases}$$

$$\text{effects} := \begin{cases} \text{at}(\mathsf{Ra}, W_1) \wedge \\ \text{energy}(\mathsf{Ra}) - = 8. \end{cases}$$

The three preconditions need to hold *before* executing the action Navigate and the two invariants need to hold *throughout* the execution of Navigate. The effect describes the change in the rover's state *after* executing the action Navigate.

In real plan execution, the aim of the monitoring module is to verify that the action is executable. A dedicated monitoring module thus checks whether the preconditions and the invariants hold in the current state. All this information is kept in the *monitor-parameters*. Formally, we define the *monitor-parameters* as a tuple $\langle \mathcal{L}, \mathcal{T}, \mathcal{V} \rangle$, where:

- $\mathcal{L}$ is a tuple $\langle L_v, L_i \rangle$ with the set of variables to be monitored (L_v) and an identifier (L_i) to express whether the variable is an invariant I, a precondition C, or an effect E, i.e., those that are directly related to the plan.
- $\mathcal{T}$ is a tuple with the time at which the variable is generated (t_g), and the earliest (t_e) and latest (t_l) time at which the variable will be used.
- $\mathcal{V}$ is the value range for each variable, denoting the set of correct values that the variables can take on.

Table 1, shows the *monitor-parameters* for the action Navigate of $\mathsf{R_a}$. For example, the last entry in the table describes the effect of executing Navigate on the energy level of the rover, i.e., the energy level will decrease by 8 units after execution.

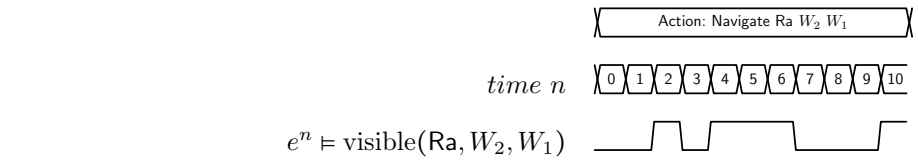
Table 1. Monitor-parameters of action Navigate

L_v	L_i	t_g	t_e	t_l	V
can_traverse($\mathsf{R_a}$,$\mathsf{W_2}$,$\mathsf{W_1}$)	I	0	0,01	10,01	true
visible($\mathsf{W_2}$,$\mathsf{W_1}$)	I	0	0,1	10,1	true
at($\mathsf{R_a}$,$\mathsf{W_2}$)	C	0	0,1	0,1	true
available($\mathsf{R_a}$)	C	0	0,01	10,1	true
energy($\mathsf{R_a}$)	C	0	0,01	0,1	$[8,\infty)$
at($\mathsf{R_a}$,$\mathsf{W_1}$)	E	10.1	10,1	20,1	true
energy($\mathsf{R_a}$) decreasing	E	10,1	10,1	10,1	8

3 From Plan Execution Monitoring to Runtime Verification

In this section, we briefly summarize the foundations of runtime verification and real-time temporal logics which we will use to express templates in our framework. For further details, we refer the reader to more elaborates sources such as [2, 5].

Real-time systems (such as the Mars Rovers) often do not only need to comply with a set of functional requirements but also – equally important – with tight timing constraints. Thus, the underlying logics need to allow to reason about certain timing assumptions. We will illustrate the concept of runtime verification along the following example:



which shows the validity of $e^n \vDash \text{visible}(\mathsf{Ra}, W_2, W_1)$, for the prefix of execution e, where $\text{visible}(\mathsf{Ra}, W_2, W_1)$ is a proposition over the state of the system. This information is incrementally collected during a run of the system, i.e., while executing the action Navigate.

Executions. Let $e = (s_t)_{t\geq 0}$ be the (sequential) execution of a set of actions $a \in \mathcal{A}$ where s_t is a state of the system at time t. A proposition $p \in \mathcal{P}$ holds on s_t iff $p \in s_t$. Denote by e^n, for $n \in \mathbb{N}_0$, the *execution prefix* $(s_t)_{0\leq t\leq n}$. For example $\mathsf{Navigate}, \mathsf{TakeImage} \in \mathcal{A}$ and $\{\text{energy}(\mathsf{Ra}) \geq 8, \text{visible}(\mathsf{Ra}, W_2, W_1), \dots\} \in \mathcal{P}$. In the running example, $\mathsf{W_1}$ is visible for $\mathsf{R_a}$ at times $n \in \{2, 4, 5, 6, 10\}$ (we write $e^n \vDash \text{visible}(\mathsf{Ra}, W_2, W_1)$), whereas $\mathsf{W_1}$ is not visible at times $n' \in \{1, 3, 7, 8, 9\}$ we write $e^{n'} \nvDash \text{visible}(\mathsf{Ra}, W_2, W_1)$).

Temporal Logics. In runtime verification a formal specification formalism is used to specify correctness claims over an execution. A popular formalism is linear temporal logic (LTL) and it's real-time extension metric temporal logic (MTL). MTL [1] extends LTL by replacing the qualitative temporal modalities of LTL by

quantitative modalities that respect time bounds. The syntax of MTL is defined as follows. For every atomic proposition $p \in \mathcal{P}$, p is a formula. $J = [t, t']$ describes a time interval for some $t, t' \in \mathbb{N}_0$. If φ and ψ are formulas, then so are:

$$\neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \varphi \; U_J \; \psi$$

For a MTL formula ξ, time $n \in \mathbb{N}_0$ and execution e, we define ξ *holds at time* n *of execution* e, denoted $e^n \vDash \xi$, inductively as follows:

$$\begin{array}{lll} e^n \vDash \mathit{true} & & \text{is } \mathsf{true}, \\ e^n \vDash \Sigma & \text{iff} & \Sigma \in AP \text{ holds in } s_n, \\ e^n \vDash \neg\varphi & \text{iff} & e^n \nvDash \varphi, \\ e^n \vDash \varphi \bullet \psi & \text{iff} & e^n \vDash \varphi \bullet e^n \vDash \psi \text{ with } \bullet \in \{\wedge, \vee, \rightarrow\}, \\ e^n \vDash \varphi \; U_J \; \psi & \text{iff} & \exists i (i \geq n) : (i - n \in J \wedge e^i \vDash \psi \wedge \forall j (n \leq j < i) : \; e^j \vDash \varphi). \end{array}$$

The Until within interval modality $\varphi \; U_J \; \psi$ allows to encode timed relationships between two formulas, with the intended meaning: φ *needs to hold continuously until (at some time within* J*)* ψ *holds.* The time bounds described by J are relative to the current time n. AP denotes the set of atomic propositions of the formula. In our case we use conjunctions of inequealities over variables to be monitored. Naturally, a formula ξ *satisfies execution* e, denoted $e \vDash \xi$, iff for all $i \in \mathbb{N}_0$, it holds that $e^i \vDash \xi$. With the dualities [4]

$$\begin{array}{ll} \mathsf{true} \; U_J \; \phi \equiv & \vec{\exists_J} \varphi \\ \neg \vec{\exists_J} \neg\phi \quad \equiv & \vec{\forall_J} \; \varphi \end{array}$$

we arrive at two additional modalities: $\vec{\exists_J}\varphi$ with the intended meaning φ *needs to hold eventually within the interval* J as well as $\vec{\forall_J} \; \varphi$ interpreted as φ *is an invariant within interval* J. Technically, a single observer capable of monitoring $\varphi \; U_J \; \psi$ is sufficient to evaluate any of $\vec{\forall_J} \; \varphi, \vec{\exists_J}\varphi$ claims, as we can always rewrite them by the equivalences above to $\varphi \; U_J \; \psi$. We argue that this fine grained breakdown of the until modality has the following advantages: (a) Allows to directly map frequently occurring claims to a modality and (b) yields more efficient observers compared to instantiating a full-fledged $\varphi \; U_J \; \psi$ observer (anonymous).

The Runtime Verification Problem. Having defined executions and temporal logics, we can define a runtime verification problem as: *Given a temporal logic formula* ξ *and an execution* e*, does* $e^n \vDash \xi$ *for* $n \in \mathbb{N}_0$ *hold?*

Monitors. Checking whether a MTL formula holds at time $n \in \mathbb{N}_0$ in some execution $e = (s_t)_{t \geq 0}$ can be determined by results from the current state s_n and it's successor states s_{n+1}. For example, evaluating the invariant $\xi = \vec{\forall}_{[4,6]} \; \text{visible}(\mathsf{Ra}, W_2, W_1)$ on execution $e = (s_t)_{t \geq 0}$ requires to check:

$$e^n \vDash \xi \Leftrightarrow \bigwedge_{t=4}^{6} (\text{visible}(\mathsf{Ra}, W_2, W_1) \text{ holds on } s_t)$$

Note that the problem of monitoring $e^n \vDash \xi$ has been studied extensively in the past; see [18] for a survey. Thus, efficient algorithms to decide $e^n \vDash \xi$ are not

an aim of this paper. For our implementation, we make use of existing algorithms. Possible choices include: a) translate the temporal formula into a finite-state automaton that accepts all the models of the specification. The translation may be bases on an on-the-fly adaption of the tableau construction [13, 21] or make use of timed automata [8, 16], b) restricting MTL to its *safety* fragment and to wait until the bounded future operators have elapsed and decide validity afterwards [17, 7] and c) restricting temporal logics to its *past-time* fragment [19, 6, 16, 8]. The conditions and effects for the action Navigate can be captured in MTL:

$$\varphi_1 := (n == 0) \rightarrow \forall^{\rightarrow}_{[0,10]}(\text{can\_traverse}(\mathsf{Ra}, W_2, W_1) \wedge \text{visible}(\mathsf{Ra}, W_2, W_1))$$
$$\varphi_2 := (n == 0) \rightarrow (\text{at}(\mathsf{Ra}, W_2, W_1) \wedge \text{energy}(\mathsf{Ra}) \geq 8 \wedge \text{available}(\mathsf{Ra}, W_2, W_1))$$
$$\varphi_3 := (n == 10) \rightarrow (\text{energy}(\mathsf{Ra})- = 8)$$

Property φ_1 captures the invariant condition for the rover execution of action Navigate. Informally, it requires that for all times $n' \in [0, 10]$ the predicates can_traverse(Ra, W_2, W_1) and visible(Ra, W_2, W_1) need to hold. Property φ_2 checks for the preconditions: At time $n = 0$, $(\text{at}(\mathsf{Ra}, W_2, W_1) \wedge \text{energy}(\mathsf{Ra}) \geq 8 \wedge \text{available}(\mathsf{Ra}, W_2, W_1))$ needs to hold. Property φ_3 checks for the effect of the action: At time $n = 10$, make sure that the energy decreases by 8 units. The force of capturing conditions and effects of an action is that the problem of plan execution monitoring can now be translated into a runtime verification problem: *Decide whether* $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ *holds on execution* e*, i.e., does* $e \vDash \varphi_1 \wedge \varphi_2 \wedge \varphi_3$ *?*

4 Template-based Synthesis of Plan Execution Monitors

In this section, we will provide a generalization of the MTL encodings of the conditions and effects associated with the action Navigate in the Mars Rover example. We give an algorithm that allows to parametrize the templates according to the monitor-parameters derived from the *decision support module*.

We start with four templates that allow to encode arbitrary monitor-parameter $\langle \mathcal{L}, \mathcal{T}, \mathcal{V} \rangle$ into an MTL specification. The first one encodes invariants and their associated time bounds whereas the second one encodes preconditions. The third one encodes the effects. And the final template the ensures that for a single action, the conjunction of all invariants and preconditions needs to hold.

Template 1: Invariants

$$\varphi_I := (n == T_a) \rightarrow \forall^{\rightarrow}_{[T_b, T_c]}(\bigwedge_{i \in T_d} \text{pred}(i))$$

Template 2: Preconditions

$$\varphi_P := (n == T_a) \rightarrow (\bigwedge_{i \in T_b} \text{pred}(i))$$

Template 3: Effects

$$\varphi_E \coloneqq ((n == T_a) \rightarrow (\bigwedge_{i \in T_b} \text{pred}(i))) \wedge ((n == T_c) \rightarrow (\bigwedge_{j \in T_d} \text{pred}(j)))$$

Template 4: Consistency

$$\varphi_C \coloneqq \bigwedge_{i \in I} \varphi_I^i \wedge \bigwedge_{j \in P} \varphi_P^j \wedge \bigwedge_{k \in E} \varphi_E^k$$

The templates from above are instantiated by Algorithm 1. The number of formulas generated is linear in the number of $\langle \mathcal{L}, \mathcal{T}, \mathcal{V} \rangle$ tuples in the monitor-parameters, i.e., six in the case of the action Navigate.

Algorithm 1 Template-based Synthesis.

Let φ_C be the resulting MTL encoding
for each $\langle \mathcal{L}, \mathcal{T}, \mathcal{V} \rangle$ in monitor parameters **do**
  if $\mathcal{L}.L_i$ is of type Invariant **then**
    $\varphi_I^i \leftarrow (n == \mathcal{T}.t_e) \rightarrow \forall^{\rightarrow}_{[0,(\mathcal{T}.t_l - \mathcal{T}.t_e)]}(\mathcal{L}.L_v)$
  end if
  if $\mathcal{L}.L_i$ is of type Precondition **then**
    $\varphi_P^j \leftarrow (n == \mathcal{T}.t_e) \rightarrow (\mathcal{L}.L_v)$
  end if
  if $\mathcal{L}.L_i$ is of type Effect **then**
    $\varphi_E^k \leftarrow (n == \mathcal{T}.t_e) \rightarrow (\mathcal{L}.L_v) \wedge (n == \mathcal{T}.t_l) \rightarrow (\mathcal{L}.L'_v)$
  end if
end for
$\varphi_C \leftarrow \bigwedge_{i \in I} \varphi_I^i \wedge \bigwedge_{j \in P} \varphi_P^j \wedge \bigwedge_{k \in E} \varphi_E^k$
return φ_C

Fig. 1 shows the integration of our template-based synthesis approach into the multi-agent framework. Inputs to the decision support module (DSM) are the generated Plan and the Domain. The DSM then derives, through an anytime regression approach, a set of monitor-parameters $\langle \mathcal{L}, \mathcal{T}, \mathcal{V} \rangle$ which are the input to the template-based synthesis algorithm. Based on templates above, this step yields the MTL encoding φ_C of the preconditions and invariants which are required to hold during execution of the plan. In the final step, we instantiate the algorithms described by [19], to derive a monitor for $e^n \vDash \varphi_C$. Please note that, the process of generating the monitor is fully automatic, with no manual intervention or modelling required.

The resulting monitor is fully executable and might run as a dedicated software task at the agent's software stack, or, alternatively be compiled into executable hardware blocks, i.e., targeting a field-programmable gate array (FPGA) or an application-specific integrated circuit (ASIC) platform. ASIC has the advantage that monitoring can be performed external to the agent, i.e., without interfering with it's runtime behavior.

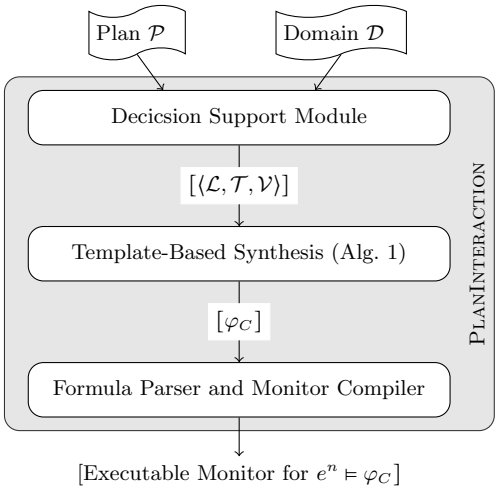


Fig. 1. Toolchain integration into the PlanInteraction framework.

5 Templates for Continuous Monitoring

We now discuss further templates which allow to encode deep relationships among physical processes of variables. We discuss several optimizations to keep the resulting monitors small.

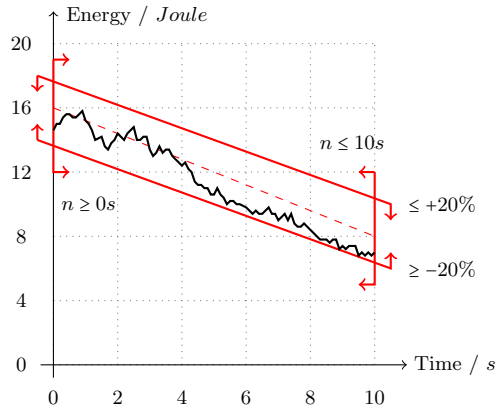


Fig. 2. Continuous monitoring the effect energy(Ra)− = 8 of action Navigate.

Consider the scenario depicted in Fig. 2, where the energy plot describes a discharge characteristic typical for the action Navigate. The exact discharge plot is different from run to run as it is is influenced by factors hidden to the

high level planner. Unforeseen changes in the terrain or sensor noise require to allow for some kind of safety margin describing the allowed range for the battery drain during movement. Therefore, a monitoring strategy which continuously evaluates energy(Ra) and compares the result against $E(n) = -0.8 \cdot n + 16$ is infeasible in a real-life setting. Ideally, we want to monitor a weaker version of $E(n) = -0.8 \cdot n + 16$, such as the two-dimensional convex polyhedron bounded by the conjunction of the following inequalities:

$$\varphi_l := \begin{cases} n \geq 0 \wedge \\ n \leq 10 \wedge \\ E(n) \geq \ 80\% \cdot (-0.8 \cdot n + 16) \wedge \\ E(n) \leq 120\% \cdot (-0.8 \cdot n + 16). \end{cases}$$

We can generalize from these constraints and derive another template that allows to encode linear decreasing or increasing relationships, such as the constraints over the energy level of the rover (Template 5). Again T_a, T_b, T_c are directly created from the monitor-parameters $\langle \mathcal{L}, \mathcal{T}, \mathcal{V} \rangle$ and predicates(T_d) selects the predicate returning the variable to be monitored. $T_f \in [0, 1]$ allows to specify the safety margin, whereas T_e is the slope of the expected increasing or decreasing characteristics and T_g is the result of evaluating predicates(T_d) at time $n == T_a$.

Template 5: Linear decreasing/increasing with Safety Margin

$$\begin{aligned} \varphi_L := (n == T_a) \rightarrow \\ \forall^{\rightarrow}_{[T_b, T_c]} (&\text{pred}(T_d) \leq (1 + T_f) \cdot (T_e \cdot n + T_g) \wedge \\ &\text{pred}(T_d) \geq (1 - T_f) \cdot (T_e \cdot n + T_g)) \end{aligned}$$

6 Conclusion and Future Work

We have studied the problem of automatically synthesizing run-time observers for plan execution monitoring from the domain description and the scheduled plan. Monitor generation works in two phases: First, we use a set of templates to automatically compile an equivalent encoding in metric temporal logic (MTL) of the constraints to be monitored. Second, we use algorithms know from the runtime verification community to compile efficient runtime monitors. The monitors are efficient in the sense that monitoring allows to include timing constraints and detects deviations from the scheduled plan as early as possible. This yields greater flexibility for a high-level re-planner module.

As future work, we plan to conduct an industrial case-study to demonstrate the benefits of our approach in real-life planning scenarios. Additionally, we plan to evaluate the use of probabilistic reasoning (for example Bayesian Networks, as in [20]) to assess the likelihood of some pre-defined error hypothesis. This would not only help us to detect deviations from the plan at mission time, but also provide reasoning about the root cause of the detected malfunction. In this setting, intermediate outputs of our runtime monitors serve as input to the reasoning module. We believe that this information would be valuable for a high-level re-planner module.

References

1. Alur, R., Henzinger, T.A.: Real-time Logics: Complexity and Expressiveness. In: LICS. pp. 390–401. IEEE (1990)
2. Alur, R., Henzinger, T.: Logics and models of real time: A survey. In: Real-Time: Theory in Practice, LNCS, vol. 600, pp. 74–106. Springer (1992)
3. Ambros-Ingerson, J.A., Steel, S.: Integrating planning, execution and monitoring. In: Proceedings of the AAAI. pp. 83–88 (1988)
4. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
5. Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G.J., Rosu, G., Sokolsky, O., Tillmann, N. (eds.): Runtime Verification - First International Conference, Proceedings, LNCS, vol. 6418. Springer (2010)
6. Basin, D., Klaedtke, F., Zălinescu, E.: Algorithms for monitoring real-time properties. In: RV. LNCS, vol. 7186, pp. 260–275. Springer (2011)
7. Basin, D.A., Klaedtke, F., Müller, S., Pfitzmann, B.: Runtime monitoring of metric first-order temporal properties. In: FSTTCS. pp. 49–60 (2008)
8. Divakaran, S., D'Souza, D., Mohan, M.R.: Conflict-tolerant real-time specifications in metric temporal logic. In: TIME. pp. 35–42 (2010)
9. Erann Gat, J.F., Miller, D.: Planning for execution monitoring on a planetary rover. In: In Proceedings of the Space Operations Automation and Robotics Workshop (1990)
10. Fikes, R.E., Hart, P.E., Nilsson, N.J.: Readings in knowledge acquisition and learning. chap. Learning and executing generalized robot plans, pp. 485–503. Morgan Kaufmann Publishers Inc. (1993)
11. Fritz, C., McIlraith, S.A.: Monitoring plan optimality during execution. In: ICAPS. pp. 144–151 (2007)
12. Gat, E., Slack, M.G., Miller, D.P., Fiby, R.: Path planning and execution monitoring for a planetary rover. In: Proceedings of the IEEE International Conference on Robotics and Automation. pp. 20–25 (1990)
13. Geilen, M.: An improved on-the-fly tableau construction for a real-time temporal logic. In: CAV. pp. 394–406 (2003)
14. Gianni, M., Papadakis, P., Pirri, F., Liu, M., Pomerleau, F., Colas, F., Zimmermann, K., Svoboda, T., Petricek, T., Kruijff, G.J., Khambhaita, H., Zender, H.: A unified framework for planning and execution-monitoring of mobile robots. In: PAMR. AAAI, AAAI Press (8 2011)
15. Kvarnström, J., Heintz, F., Doherty, P.: A temporal logic-based planning and execution monitoring system. In: ICAPS. pp. 198–205 (2008)
16. Maler, O., Nickovic, D., Pnueli, A.: Real time temporal logic: Past, present, future. In: FORMATS. pp. 2–16 (2005)
17. Maler, O., Nickovic, D., Pnueli, A.: On synthesizing controllers from bounded-response properties. In: CAV. pp. 95–107 (2007)
18. Maler, O., Nickovic, D., Pnueli, A.: Checking temporal properties of discrete, timed and continuous behaviors. In: Pillars of Computer Science. pp. 475–505. Springer (2008)
19. Reinbacher, T., Függer, M., Brauer, J.: Real-time runtime verification on chip. In: RV. LNCS, vol. 7687, pp. 110–125. Springer (2012)
20. Schumann, J., Mengshoel, O.J., Srivastava, A.N., Darwiche, A.: Towards software health management with Bayesian networks. In: FoSER. pp. 331–336 (2010)
21. Thati, P., Roşu, G.: Monitoring Algorithms for Metric Temporal Logic specifications. ENTCS 113, 145–162 (2005)