2019 IEEE/ACM 2nd International Workshop on Robotics Software Engineering (RoSE)

MAPmAKER: Performing Multi-Robot LTL Planning Under Uncertainty

Sergio García*, Claudio Menghi†, and Patrizio Pelliccione*‡
*Chalmers | University of Gothenburg, Gothenburg (Sweden)
† University of Luxembourg, Luxembourg City (Luxembourg)
‡ University of L'Aquila, L'Aquila (Italy)
Email: sergio.garcia@gu.se, claudio.menghi@uni.lu, patrizio.pelliccione@gu.se

***Abstract*—Robot applications are being increasingly used in real life to help humans performing dangerous, heavy, and/or monotonous tasks. They usually rely on planners that given a robot or a team of robots compute plans that specify how the robot(s) can fulfill their missions. Current robot applications ask for planners that make automated planning possible even when only *partial knowledge* about the environment in which the robots are deployed is available. To tackle such challenges we developed MAPmAKER, which provides a decentralized planning solution and is able to work in partially known environments. Decentralization is realized by decomposing the robotic team into subteams based on their missions, and then by running a classical planning algorithm. Partial knowledge is handled by calling several times a classical planning algorithm.**

Demo video available at: https://youtu.be/TJzC_u2yfzQ

I. Introduction

Robotic applications usually rely on a set of robots that are used to perform missions. The term mission can refer to a *global mission*, i.e., the high-level mission that must be accomplished by the whole team [1] or a *local mission*, i.e., the mission that should be achieved by a single robot, possibly by collaborating with other robots [2]. Planners are one of the main ingredients that allow robots to achieve missions. A *planner* is a software component that receives as input a model of the robotic application and computes a set of actions—a *plan*—that, if performed, allow the achievement of a desired mission [3]. Recent works in robotics have defined robot applications using finite transition systems and some of them define their local missions as a Linear-time Temporal Logic (LTL) property (e.g., [2], [4]–[6]). Current robotic applications require planners to address two main challenges: 1) the planning algorithm should work when (only) partial knowledge about the system—including the robots and their working environment—is present; 2) the planning problem should be solved by decentralized algorithms that help to reduce the planning overhead.

Several works studied centralized planners that are able to manage *teams* of robots that collaborate to achieve a certain goal (a global mission) [1], [7]. However, planning is computationally expensive, especially when the number of robots within the team increases and they need to collaborate to fulfill their local missions. For this reason, research interest had focused on decomposing a global mission into a set of local missions to be achieved by each robot of the team [2], [5], [8]. These local missions have been recently exploited by *decentralized* planners [2], i.e., planners that instead of evaluating the global mission over the whole team of robots, analyze the satisfaction of local missions inside a subset of the team of robots. In this way, the problem of finding a collective team behavior is decomposed into sub-problems that avoid the expensive fully centralized planning. However, the applicability of these algorithms has never been studied when only partial knowledge about the system is available.

The role of partial knowledge or uncertainty in software development has been strongly studied in literature. Research has been done on how to consider partial knowledge in requirement analysis and elicitation [9]–[11], in the development of a model of the system that satisfies a set of desired properties [12]–[16], and in checking whether an already designed model possesses some properties of interest [17]–[19]. However, most of the existing planners assume that the environment in which the robots are deployed is known [20]. This assumption does not usually hold in real-word scenarios [21], where, for example, the robots navigate in environments affected by natural disasters, where the movement between locations or the execution of specific actions may be impossible. While planners that consider partial information about the environment in which the robots operate exist (e.g., [22]–[24]), they usually rely on probabilistic algorithms and are not *decentralized*.

This work presents MAPmAKER: a Multi-robot plAnner for PArtially Known EnviRonments. MAPmAKER provides a *decentralized* planning solution that works in *partially known* environments. Decentralization is realized by decomposing the robotic team into subteams based on their missions, and then by running a classical planning algorithm. Partial knowledge is handled by calling twice a classical planning algorithm. The theory that supports MAPmAKER including proofs of correctness, a detailed description of the modelling formalisms, and the verification procedures can be found in [4]. In this paper we present the implementation of MAPmAKER, the components that compound it, the models it uses, and how it can be used. We also provide a demonstration video to illustrate such concepts. In this sense, the contribution is more a proof of concept than a tool ready to be used for real-world scenarios. MAPmAKER builds upon the planner proposed by Tumova et al. [2] and is evaluated by analyzing its behavior on a robot application simulating a hospital environment with

978-1-7281-2249-6/19/$31.00 ©2019 IEEE
DOI 10.1109/RoSE.2019.00008

the presence of uncertainty. MAPmAKER together with 1) a complete replication package, 2) a set of videos showing MAPmAKER's performance solving the scenario presented at the previous bullet, and 3) a brief user guide that defines the main functionalities are available at our Github repository [25].

II. MAPmAKER's Overview

An overview of MAPmAKER is shown in Fig. 1. The MAPmAKER's planner takes as input the models of the robots (①), of the environment in which they are deployed (②), and of each robot's mission (③). Both the models of the robots and their environment may be partial since they may contain information uncertainty. The implemented planner is able to compute plans that definitely ensure the mission satisfaction, i.e., definitive plans (④), and plans that may ensure property satisfaction since they depend on some partial knowledge present in the models of the robots and the environment (⑤). More precisely, a *definitive plan* is a sequence of actions (e.g., move from *a* to *b*) that ensure the satisfaction of the local mission for each robot. A *possible plan* is a sequence of actions that may satisfy the local mission due to some unknown information about the model of the robots or the environment in which they are deployed. If MAPmAKER is not able to find neither a definitive nor a possible plan a message is sent to the user (⑥). Otherwise, the *Plan selector* chooses between definitive and possible plans (if both are present) or chooses the possible plan if no definitive plan is present. Definitive plans are not present when the only way to satisfy the local mission is based on some unknown information about the model of the robots or the environment in which they are deployed. MAPmAKER then executes the selected plan (⑦).

As robots perform plans, information about uncertain parts of the model is detected. MAPmAKER updates the models with the detected information (⑧) and if it detects that a plan is not anymore executable, the planner is re-executed (⑨).

In the following, we provide some additional information about the inputs processed by MAPmAKER, the planning algorithm, the selection between definitive and possible plans and how models are updated when information about uncertain parts is detected.

Models of the robots and their environment. The models of the robots and their environments are provided using a specific form of transition system that allows the specification of uncertain parts; further information might be found in [4]. There is one `Robotx.m` (e.g., `Robot1`) file for each robot in the global mission, containing a function that generates the robot's model. Each file contains information about the robot, like its `id`, the atomic prepositions of the LTL formula it may perform, its initial position, services provided by the robot, etc. The `MissionRobotx.m` file contains a correspondent function for each robot model, which encodes the number of actions the robot must perform and whether it is required that other robots must help it to accomplish certain actions and returns an automaton corresponding with an LTL formula (the transitions within this file describe the formula). The environment is

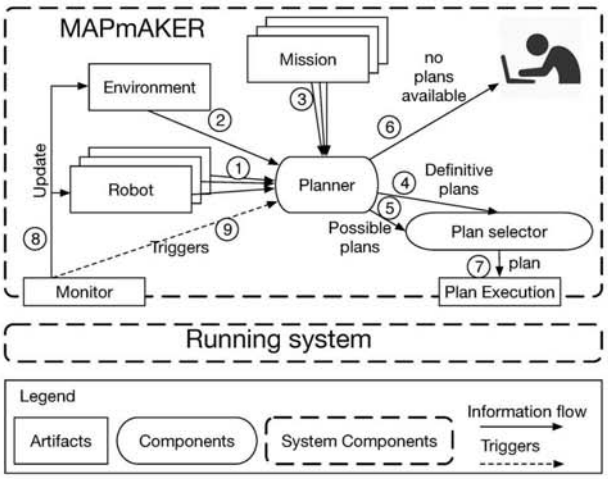


Fig. 1: Overview of MAPmAKER.

defined as a grid where transitions between its conforming cells may or may not be possible. This information is encoded in two environment models: one that contains uncertainty e.g., `EnvironmentMap.m` and one that does not (e.g., `RealEnvironmentMap.m`). For more technical details, the interested reader can refer to the provided repository [25].

The proposed models embed partial knowledge as follows:
- *Partial knowledge about the actions execution.* The execution of some actions is uncertain, meaning that it is unclear whether an action can be executed. This type of partial knowledge allows specifying that the transition between two of the cells of the grid map of the environment (see Fig. 2) can be: always possible, always impossible (i.e. a wall), or not known (i.e. a door between two rooms that can be open or closed).
- *Unknown service provisioning.* It is unclear whether a service—i.e., "events of interest associated with execution of certain actions rather than over atomic propositions" [5]—can be provided in a specific location. For example, it is unclear whether a robot can take a picture in a given map location. This uncertainty may be caused for example by the presence of an object that covers the robot visual in that location.
- *Unknown meeting capabilities.* Robots can meet and synchronize in certain locations. For example, it is unclear whether two robots can exchange a load in a given map location. This uncertainty may be caused by a collapsing registered in the environment where the robots are deployed.

Our tool may be classified at level 2 using the taxonomy presented in [26] as when uncertainty is detected, during runtime the system adapts and computes a new plan.

Mission specification. Each robot is able to perform a complex mission, which is specified using an LTL formula. This formula specifies how the services must be provided by the robots. For example, a mission for a robot r_1 may require r_1 to periodically load debris on r_2. Thus, in order to allow robot r_1 to fulfill its mission, it is necessary that robots r_1 and r_2 synchronize their behaviours.

Planning. The *Planner* uses the models of the robot(s)

and the environment to compute plans that allow satisfying the missions of the robots. It distributes the robots of the robotic application into subteams (i.e., "dependency classes") based on the mission that each robot has to achieve. Each dependency class contains a subset of robots that depend on each other for achieving their missions. After dependency classes are computed they are considered in isolation regarding the computation of plans.

To compute a plan for a dependency class the LTL formulae that are used to describe missions are evaluated on partial models. Possible and definitive plans are computed by executing a classical planning algorithm twice: once for computing possible plans and once for computing definitive plans.

Choosing between definitive and possible plans. The *Plan selector* chooses between possible and definitive plans. Several policies can be applied to choose between these plans. Possible plans can be chosen only in cases in which a definitive plan is not present. Another policy may choose the plan with the shortest length, or it may consider non-functional aspects of the plans e.g., time to perform certain actions, or likelihood of detecting true or false evidence about partial information. In this work we assume that the planner always chooses the shortest length plan. This policy may reduce energy consumption.

Detection of uncertain information. As robots perform actions and navigate within the environment, information regarding uncertain services and meeting capabilities can be detected. Specifically, robots detect whether actions, services, and meeting capabilities are executable, provided, and possible, respectively. MAPmAKER updates the models of the robots and of the environment with the detected information. Then, if needed, the planning algorithm is triggered and re-executed.

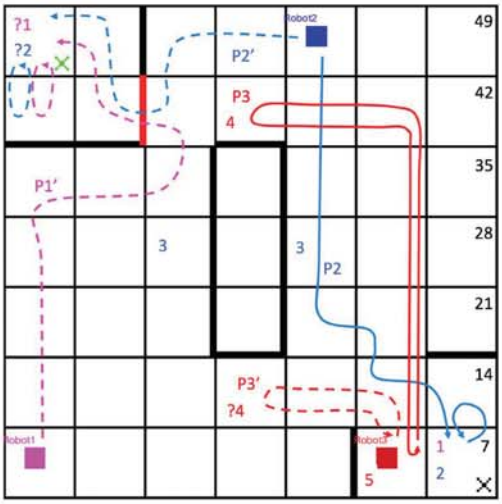
III. MAPmAKER in Action

MAPmAKER is developed as a MATLAB [27] standalone application. It is developed on top of an existing planner—presented in [2]—which has been chosen since it already implements a decentralized planning procedure. MAPmAKER calls this planner twice considering two different versions of the model of the robots and their environments. The results obtained by performing this procedure are sound and correct. Additional details and proofs can be found in [4].

```
mapmakerRunner(robots, environment , missions);
mapmaker_exp('Scenario', 'Experiment', 'Location')
```

Listing 1: Running MAPmAKER

MAPmAKER can be executed in two ways, as shown in Listing 1. The first option is used to compute plans for custom models of mission, environment, and robots. `robots` is a variable that specifies the number of robots and their models. `environment` is a model of the environment and its uncertainty. `missions` contains the local mission to be achieved by each robot. With the second option, we provide a way of replicating the experiments presented in this paper and in [4]. `Scenario` is a `.m` file containing the model of the environment and the robots. `Experiment` encodes each robot's mission. `Location` defines where the experiments are allocated—i.e., RQ1, RQ2, or RunningExample.



$\phi_1 = \mathsf{G}(\mathsf{F}(\mathit{help\_grasping}))$: r_1 periodically helps $r2$ with grasping (service *help_grasping* associated with action 1).
$\phi_2 = \mathsf{G}(\mathsf{F}(\mathit{fetch\_supplies} \wedge \mathsf{F}(\mathit{deliver})))$: r_2 repeatedly fetches supplies (service *fetch_supplies*, action 2) and delivers them (service *deliver*, action 3).
$\phi_3 = \mathsf{G}(\mathsf{F}(\mathit{take\_snapshot} \wedge \mathsf{F}(\mathit{send\_info})))$: r_3 repeatedly takes pictures (service *take_snapshot*, action 4) and sends them using the communication network (service *send_info*, action 5).

Fig. 2: MAPmAKER usage scenario.

MAPmAKER shows a graphical interface as the one presented in Fig. 2 when executed. Fig. 2 is a screenshot of MAPmAKER's performance where we changed the size of some numbers, added plans, and labeled some cells. The grid represents the environment in which the robots move. Each cell represents a location of the environment and has a number associated. Robots are represented by colored squares. Actions are used to encode movements and skills of the robots (i.e. labeled actions from 1 to 5). A robot cannot move between adjacent cells if they are separated by thick bordered lines. Whenever it is unclear if a robot can move between adjacent cells, these cells are separated by a red border. Whenever a service can be provided by a robot in a cell, the cell is labeled with the associated action and the color of the corresponding robot (a question mark precedes the action number if the provision is unclear). Synchronization capabilities are represented by a black cross (which becomes green if is unclear whether two robots can synchronize in a cell).

Assume that the robots r_1, r_2, and r_3 have the *local missions* defined in Fig. 2. In a hospital environment, r_1 is a dexterous mobile manipulator that aims at helping r_2, a heavy-duty mobile manipulator, with grasping certain medical supplies. It means that r_1 and r_2 must meet in cells where service *fetch_supplies* is provided. Once these supplies are grasped, r_2 brings them to the surgery table in the middle of the room. In the meanwhile, r_3 takes pictures of the table and sends them

using the communication network.

The same figure also shows different plans that can be performed by the robots, accomplishing a number of actions in a periodic fashion. Every robot can perform different plans, but only some of them are showed to improve readability. P1' represents a possible plan where true evidence was detected by the robot (in actions execution, service provisioning, and meeting capabilities), reaching the cell to accomplish service *help_grasping* (action 1) and meet with r_2. P2 is a definitive plan, where r_2 must synchronize with r_1 to accomplish service *fetch_supplies*. P2' represents a possible plan (same uncertainty seen in P1') that performs similar actions to P2 but in another room. P3' represents a possible path where robot r_3 accomplishes action 4 in cell 11 and action 5 in cell 6. Finally, P3 is a definitive plan, which substitutes the location of service *take_snapshot* from cell 11 to cell 39.

IV. Evaluation

To evaluate MAPmAKER we formulate a research question, **RQ**: *Is MAPmAKER able to perform planning in partially known environments?* To answer it, we had considered the simulated scenario introduced in Sec. III. We created a partial robot application starting from the models of the robots and their environment. We then introduced uncertainty in the three considered dimensions introduced in Sec. II. Examples of such uncertainties are whether the system has certain knowledge about the transition through doors (e.g., the one between cells 37 and 38 in Fig. 2) or about the provision of services (e.g., *deliver* at cells 24 and 26). We introduced uncertainty through a random process and created three different scenario configurations based on the same environment. We also randomized the initial position of each robot, creating three different sets of initial configurations. The nine experiments we performed to validate MAPmAKER consist of the nine possible combinations of the scenario and initial configurations.

The results show that the decentralized algorithm actually helps in improving performances and that MAPmAKER is able to compute plans in situations where traditional planners cannot. MAPmAKER also improves the performance in terms of plan length in various situations. In the folder `ResultsPaperRoSE` of our repository [25] we provide a set of videos showing the performance of MAPmAKER in these experiments. We also provide results, containing computation time, plan length, false and true evidences found by the robots, and ratio between the definitive and possible plans in terms of computation time and plan length. The evaluation of the underlying algorithms might be found in [4].

V. Conclusion

We presented MAPmAKER, a decentralized planner for partially known environments. The MAPmAKER implementation relies on a naive implementation of a planner that comes from literature and has been customized within the proposing framework. Our evaluation showed how MAPmAKER improves planning in cases in which partial information is present.

As future work we plan to experiment in complex scenarios and with real robots. We will make use of more efficient planners to speed up the computation. Other work will include the study of appropriate policies to select between definitive and possible plans.

Acknowledgements

This work was supported by the EU H2020 Research and Innovation Programme under GA No. 731869 (Co4Robots).

References

[1] M. M. Quottrup, T. Bak, and R. Zamanabadi, "Multi-robot planning: A timed automata approach," in *ICRA*, vol. 5. IEEE, 2004.

[2] J. Tumova and D. V. Dimarogonas, "Multi-agent planning under local LTL specifications and event-based synchronization," *Automatica*, 2016.

[3] J.-C. Latombe, *Robot motion planning*. Springer, 2012, vol. 124.

[4] C. Menghi, S. García, P. Pelliccione, and J. Tumova, "Multi-robot LTL planning under uncertainty," in *FM2018*. Springer, 2018, pp. 399–417.

[5] M. Guo and D. Dimarogonas, "Multi-agent plan reconfiguration under local LTL specifications," *The Int. Journal of Robotics Research*, 2015.

[6] C. Menghi, C. Tsigkanos, T. Berger, P. Pelliccione, and C. Ghezzi, "Property specification patterns for robotic missions," in *Proc. of the 40th International Conference on Software Engineering: Companion Proceeedings*, ser. ICSE '18. ACM, 2018, pp. 434–435.

[7] M. Kloetzer, X. C. Ding, and C. Belta, "Multi-robot deployment from LTL specifications with reduced communication," in *CDC*. IEEE, 2011.

[8] P. Schillinger, M. Bürger, and D. Dimarogonas, "Decomposition of finite LTL specifications for efficient multi-agent planning," in *DARS*, 2016.

[9] C. Menghi, P. Spoletini, and C. Ghezzi, "Integrating goal model analysis with iterative design," in *REFSQ*. Springer, 2017.

[10] ——, "COVER: Change-based goal verifier and reasoner," in *REFSQ Workshops*, 2017.

[11] E. Letier, J. Kramer, J. Magee, and S. Uchitel, "Deriving event-based transition systems from goal-oriented requirements models," *Automated Software Engineering*, vol. 15, no. 2, 2008.

[12] S. Uchitel, G. Brunet, and M. Chechik, "Synthesis of partial behavior models from properties and scenarios," *TSE*, vol. 35, no. 3, 2009.

[13] S. Uchitel, D. Alrajeh, S. Ben-David, V. Braberman, M. Chechik, G. De Caso, N. D'Ippolito, D. Fischbein, D. Garbervetsky, J. Kramer, *et al.*, "Supporting incremental behaviour model elaboration," *Computer Science-Research and Development*, vol. 28, no. 4, 2013.

[14] M. Famelis, R. Salay, and M. Chechik, "Partial models: Towards modeling and reasoning with uncertainty," in *ICSE*, 2012.

[15] A. Albarghouthi, A. Gurfinkel, and M. Chechik, "From under-approximations to over-approximations and back," in *TACAS*, 2012.

[16] A. Bernasconi, C. Menghi, P. Spoletini, L. D. Zuck, and C. Ghezzi, "From model checking to a temporal proof for partial models," in *Software Engineering and Formal Methods*. Springer, 2017.

[17] C. Menghi, P. Spoletini, and C. Ghezzi, "Dealing with incompleteness in automata-based model checking," in *Formal Methods*, 2016.

[18] G. Bruns and P. Godefroid, "Model checking partial state spaces with 3-valued temporal logics," in *CAV*, 1999.

[19] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel, "Multi-valued symbolic model-checking," *ACM TOSEM*, 2004.

[20] A. G. Cunningham, E. Galceran, R. M. Eustice, and E. Olson, "Mpdm: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving," in *ICRA*, 2015.

[21] M. Lahijanian, M. Maly, D. Fried, L. Kavraki, H. Kress-Gazit, and M. Vardi, "Iterative temporal planning in uncertain environments with partial satisfaction guarantees," *IEEE Transactions on Robotics*, 2016.

[22] N. Roy, G. Gordon, and S. Thrun, "Planning under uncertainty for reliable health care robotics," in *Field and Service Robotics*. Springer, 2006.

[23] N. E. Du Toit and J. W. Burdick, "Robot motion planning in dynamic, uncertain environments," *IEEE Transactions on Robotics*, 2012.

[24] J. F. Diaz, A. Stoytchev, and R. C. Arkin, "Exploring unknown structured environments." in *FLAIRS Conference*. AAAI Press, 2001.

[25] Repository, https://github.com/claudiomenghi/MAPmAKER/, 2017.

[26] D. M. Berry, B. H. Cheng, and J. Zhang, "The four levels of requirements engineering for and in dynamic adaptive systems," in *REFSQ*, 2005, p. 5.

[27] MATLAB, https://mathworks.com/products/matlab.html, 2017.