Model-Based Testing of an Industrial Multi-Robot Navigation System

(Extended Abstract)

Clemens Mühlbacher and Gerald Steinbauer
Institute for Software Technology, Graz University of Technology
Graz, Austria
{cmuehlba, steinbauer}@ist.tugraz.at

Stefan Gspandl and Micheal Reip
incubedIT
Hart bei Graz, Austria
{s.gspandl, m.reip}@incubedit.com

Keywords

model-based testing; multi-robot navigation; test-case coverage

1. MOTIVATION

Nowadays multi-robot system getting more and more deployed in industrial settings. The usage of such a system is motivated by the fact that it allows flexible solutions which scale well with the current work load. One example of such a multi-robot system is a fleet of transport robots in a warehouse. The use of planning and scheduling algorithm allow such a fleet to maximize the throughput in the warehouse with a minimum of interference. To maximize the utility of such systems they have to operate on a 24/7 basis. Therefore, these systems need to show a high degree of dependability.

The multi-robot system we are addressing in this work is motivated by a real industrial use case of a fleet of transport robots in a warehouse. The robots execute transportation tasks which are assigned to them in an automated way. To perform this transportation task without interference within the fleet and to respect operational restrictions of the environment a hierarchical planning approach is used [5]. The approach uses traffic areas which are defined by the user to model the allowed behavior of a robot within a particular area of the environment. This is done by treating areas like shared resources with different options for their utilization.

A proper functioning of the navigation component is crucial for the dependability of the complete robotic fleet. Thus to ensure the high degree of dependability one needs to take proper measurements during the complete life cycle of this component. One way which allows to treat all phases of the development in a unified way and offers the possibility of a high degree of automation is model-driven engineering [2]. To apply the idea of model-driven engineering to autonomous robots one can follow the idea outlined in [7]. The authors propose to use model-based techniques during the life cycle of the robot ranging from model-based testing to model-based supervision. With the help of model-based techniques tedious tasks such as testing and supervision can be automatized. In this paper we will outline how to address the problem of properly testing the system using a model-based testing approach [9, 10].

In order to show how to model the robotic fleet and how to derive a test case from this model we will use a simple running example which is a simplified version of the robotic fleet from the warehouse. The environment of the robotic fleet is divided into several convex areas which are disjoint. Each of this traffic areas may be of a certain combination of types which restrict its utilization. The following types of a traffic area are possible: (1) forbidden area, (2) one-way area, and (3) single robot area. An area which is a forbidden area forbids any robot to be within this area. A one-way are restrict the direction to moving within this area to one predefined direction. A single robot area allows only one robot to be within this area at any time.

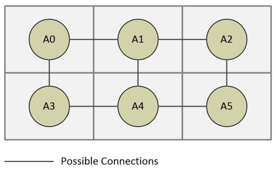
To ensure that the robotic fleet traverses the environment efficiently while respecting the restrictions of the areas a central server is used. This central server distributes the current environment map and manages reservations of robots for certain areas. These reservations are used to coordinate the robotic fleet by enforcing that the robot needs to reserve every area it traverses. In the navigation scenario the robot needs to properly move from a given start location to a defined end location. This is done by considering the reservations and the environment map. As the robotic fleet should operate efficiently each robots needs to plan the fastest path through the environment respecting all the reservations. Thus the robot may need to take a detour if it is faster than waiting until it is allowed to enter a certain area.

2. MODEL-BASED TEST CASE GENERATION

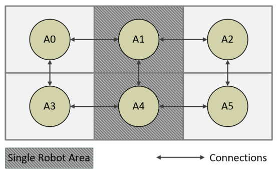
As we want to automatically derive test cases we will use model-based testing. This requires that we have a model of the desired behavior of the robotic fleet and its environment. As a model we use a set of constraints $\mathbb{C}_{DOMAIN}$. The set is the composition of three different sets of constraints $\mathbb{C}_{DOMAIN} := \mathbb{C}_{MAP} \cup \mathbb{C}_{PATH} \cup \mathbb{C}_{RESERVE}$.

$\mathbb{C}_{MAP}$ describes the environment of the robot in a topological way. The environment consists of n areas $\mathcal{A} := \{A_1, A_2, ..., A_n\}$, which are vertices in a labeled directed graph $\mathcal{G} := (\mathcal{A}, \mathcal{E})$. The set $\mathcal{E}$ describe the set of edges in the graph and describe the possible transitions between areas. As each area in the environment has a subset of types as-

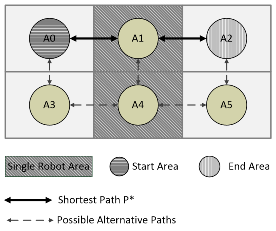
Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*



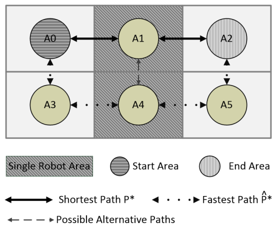
(a) Graph template used for the test case generation



(b) Generated environment for the test case



(c) Generated start and end area for the test case. Together with the shortest path



(d) Generated test case with shortest and fastest path. The test case criteria impose a detour.

Figure 1: Test case generation

signed to it we use the function $type : \mathcal{A} \rightarrow 2^{\mathcal{T}}$ to represent this mapping. Besides the constraints which are imposed by the type we need to impose constraints such that the graph can be drawn on a plain using only convex polygons. A graph which can be drawn on a plan needs to be at least planar [8] but with the additional restriction to convex polygons further constraints need to be applied [3].

$\mathbb{C}_{PATH}$ describes the path the robot takes through the environment. A path is defined as a sequence of area transitions (edges) in $\mathcal{G}$: $P = \{\langle a_s^0, a_e^0\rangle, \langle a_s^1, a_e^1\rangle, ..., \langle a_s^k, a_e^k\rangle\}$ with $\langle a_s^i, a_e^i\rangle \in \mathcal{E}$ and $\forall i = 0...k-1, a_e^i = a_s^{i+1}$. As the robot needs to find the fastest path one assigns to each path a traversal time through the function $travel(P) = \sum_{i=0...k-1} leave(a_s^i, a_e^i) + wait(a_s^i, a_e^i) + enter(a_s^i, a_e^i)$. $leave(a_s^i, a_e^i)$ defines the time it takes to leave area a_s^i towards area a_e^i. $wait(a_s^i, a_e^i)$ defines the time the robot needs to wait till it can enter the area a_e^i from area a_s^i. $enter(a_s^i, a_e^i$ defines the time it takes to enter area a_e^i from area a_s^i. As the robot needs to move as fast as possible we use the fastest path defined as $P^*.\forall P \neq P^* \rightarrow travel(P^*) \leq travel(P)$.

$\mathbb{C}_{RESERVE}$ describes the constraints imposed by the reservations done by other. This is represented by the set $\mathcal{I}$ which describes which intervals are reserved by another robot. Each interval is a tuple $I := \langle a, t_s, t_e, r\rangle$ where a represents the reserved area, t_s describes the start time, t_e the end time, and r the reserving robot. To describe the constraints on the usage of the different traffic areas with respect to the type of the area we use Allen's interval algebra [1].

Beside the domain of the robot $\mathbb{C}_{DOMAIN}$ we use an additional set of constraints $\mathbb{C}_{CRITERIA}$ to describe which test case should be derived. This description allows to steer the test case generation into a certain direction. One could for example derive a test case which forces that the path starts in an area of type 'single robot' through the constraint $\mathcal{T}_{singlerobot} \in type(a_s^0)$.

Combining the two constraint sets $\mathbb{C}_{DOMAIN}$ and $\mathbb{C}_{CRITERIA}$ the test case can be derived automatically by solving the resulting CSP. Unfortunately, due to the complexity of the resulting constraint problem one cannot use a standard constraint solver to derive a test case. Instead we split up the problem into three steps and solve the subproblem with a standard constraint solver separately. The method to derive a test case is depicted in Figure 1. In order to ensure that the graph can be realized in an environment we use a seed graph (Figure 1(a)). This graph specifies which areas and connections are possible. The environment is now derived by assigning to each area a type and choosing the connections according to the constraints of the environment (Figure 1(b)). After creating the environment, a start and end area is determined such that the constraints from the test case criteria which concern the area traversal are fulfilled (Figure 1(c)). Finally, the intervals are generated to ensure that the test case criteria are fulfilled (Figure 1(d)). If one of the steps fails a back tracking is performed and an alternative solution from the previous step is used to derive the test case.

3. EVALUATION AND CONCLUSION

We implemented the outlined system using the programming language Java and the CSP solver library Choco [6]. The test case generation for the experiments ran on an i5 with 8GB of RAM running Ubuntu 14.04 and Java 1.7. The generated test cases where used to test the multi-robot navigation system proposed in [5]. To perform this tests a test suite comprises 14 test cases where the generation took 1.6 seconds on average. Using this test suite 68.14 % of the lines of code of the complete navigation system were covered. Moreover, 31.07 % of the branches of the complete navigation system were tested. Most of the not covered lines of code and branches were either used to cover cases of dynamics in the environment which we have ignored for simplicity during the test case generation or were used to cope with faults in the configuration or communication with other software components.

To further check if the generated test cases can detect a fault we created 92 mutated versions of the navigation system following the idea of mutation testing [4]. 98.92 % of these mutated versions were correctly identified to be faulty. Thus the test suite has a high chance to detect real implementation flaws. Additionally, during the execution one flaw in the implementation was found causing an illegal memory access.

With the method outlined in this paper it is possible to automatically generate a test suite which tests the navigation system of a fleet of transport robots. Due to a divide and conquer approach the generation of the test cases is done in reasonable time allowing to simply create new tests if necessary. To steer the test case generation, the method uses a test case criterion which is described with the help of constraints in an abstract manner. For simplicity the method only specifies the behavior of the system for a static environment. To extend the method to dynamic changes is left for future work.

Acknowledgments

This work is supported by the Austrian Research Promotion Agency (FFG) under grant 843468.

REFERENCES

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] D. Brugali and A. Shakhimardanov. Component-Based Robotic Engineering (Part II) - Systems and Models. *Robotics Automation Magazine, IEEE*, 17(1):100–112, 2010.

[3] A. L. Buchsbaum, E. R. Gansner, C. M. Procopiuc, and S. Venkatasubramanian. Rectangular layouts and contact graphs. *ACM Transactions on Algorithms (TALG)*, 4(1):8, 2008.

[4] R. G. Hamlet. Testing programs with the aid of a compiler. *IEEE Transactions on Software Engineering*, SE-3(4):279–290, July 1977.

[5] S. Imlauer, C. Mühlbacher, G. Steinbauer, M. Reip, and S. Gspandl. Hierarchical planning with traffic zones for a team of industrial transport robots. In *ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP-2016)*, pages 57–64, 2016.

[6] C. Prud'homme, J.-G. Fages, and X. Lorca. *Choco Documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016.

[7] G. Steinbauer and C. Mühlbacher. Hands Off - A Holistic Model-Based Approach for Long-Term Autonomy. In *ICRA Workshop on AI for Long-term Autonomy*, 2016.

[8] R. J. Trudeau. *Introduction to graph theory.* Courier Corporation, 2013.

[9] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[10] M. Utting, A. Pretschner, and B. Legeard. A taxonomy of model-based testing approaches. *Softw. Test. Verif. Reliab.*, 22(5):297–312, Aug. 2012.