2019 IEEE International Conference on Artificial Intelligence Testing (AITest)

Constraint-Based Testing of An Industrial Multi-Robot Navigation System

Clemens Mühlbacher
Institute for Software Technology
Graz University of Technology
Graz, Austria
cmuehlba@ist.tugraz.at

Gerald Steinbauer
Institute for Software Technology
Graz University of Technology
Graz, Austria
steinbauer@ist.tugraz.at

Michael Reip
incubed IT
Hart bei Graz, Austria
m.reip@incubedit.com

Stephan Gspandl
incubed IT
Hart bei Graz, Austria
s.gspandl@incubedit.com

***Abstract*—Intelligent multi-robot systems get more and more deployed in industrial settings to solve complex and repetitive tasks. Due to safety and economic reasons they need to operate dependably. To ensure a high degree of dependability, testing the deployed system has to be done in a rigorous way.**

Advanced multi-robot systems show a rich set of complex behaviors. Thus, these systems are difficult to test manually. Moreover, the space of potential environments and tasks for such systems is enormous. Therefore, methods that are able to explore this space in a structured way are needed. One way to address these issues is through model-based testing.

In this paper we present an approach for testing the navigation system of a fleet of industrial transport robots. We show how all potential environments and navigation behaviors as well as requirements and restrictions can be represented in a formal constraint-based model. Moreover, we present the concept of coverage criteria in order to handle the potentially infinite space of test cases. Finally, we show how test cases can be derived from this model in an efficient way. In order to show the feasibility of the proposed approach we present an empirical evaluation of a prototype implementation using a real industrial use case.

***Index Terms*—constraint-based testing, constraint-satisfaction, multi-agent systems, navigation**

I. Introduction

Intelligent multi-robot systems nowadays are getting more and more deployed in industrial settings. They allow flexible solutions that scale according to the current workload. An example for such a setting is a fleet of robots operating in a warehouse to transport goods automatically. By using advanced scheduling and planning algorithms the robots are able to optimize their throughput as well as minimize interference or other criteria e.g. energy consumption. Such systems have to operate on a 24/7 basis to maximize their utility. Therefore, dependability of the systems plays a big role and nearly no human intervention should be necessary during operation.

In order to ensure the necessary high degree of dependability and therefore autonomy of such multi-robot systems proper measures should be taken during the entire life cycle of the system. It ranges from requirement engineering during the design over sound testing of hardware, software and the behavior during the implementation to supervision during operation. In order to treat all phases in a unified and highly automated way model-driven engineering [3] became popular in the last decade. In [16] the authors presented a concept based on model-based techniques that is able to support dependability for a team of autonomous robots during the entire life cycle. Model-based techniques are of particular interest in this context because they allow for a high level of automation of tedious activities such as testing and supervision.

The motivation for the work presented in this paper originates from a real industrial use case of a fleet of autonomous transport robots used in warehouse automation. The robots are able to perform transport tasks autonomously and independently while tasks are assigned to individual robots by a central server. In order to minimize interference between robots during navigation and to respect operational restrictions of the environment such as one-ways, the system uses the hierarchical planning approach presented in [6]. The approach allows the user to define traffic areas with various properties the robots have to respect during planning and execution of their navigation. Areas are treated as shared resources with different options for utilization.

The proper function of the navigation system is crucial for the dependability of the system. Therefore, the planning and execution of the navigation needs to be tested carefully. Due to the complex interplay of the robots during planning and execution as well as the potentially infinite number of different environments sound manual testing is tedious if not infeasible. Thus automating testing is preferred.

In order to tackle the problem of generating appropriate test cases for the navigation system we propose a model-based testing (MBT) approach [19], [20]. The approach presented in this paper allows for specifying all potentially desired behaviors of the navigation system as well as all potential environments in a formal model based on a constraint satisfaction problem. Because generating and executing all possible test cases encoded in the model is impossible we limit the test case generation to test cases that cover particular aspects of the system. The limitation is done by using test case criteria that are constraints and represent a declarative description of test cases. The test case criteria together with the formal description of the environment and behavior can be used to generate expressive test cases automatically.

The major contributions of the work presented in this paper

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

978-1-7281-0492-8/19/$31.00 ©2019 IEEE
DOI 10.1109/AITest.2019.00015

are (a) a concept for obtaining a formal model of the behavior of the navigation of a multi-robot system that is suitable for MBT and (b) a feasible practical implementation of an automated test case generation for a real industrial use case.

The remainder of the paper is organized as follows. In the next section we will briefly discuss the industrial use case. In Section III we will present a simplified version of the system which serves as a running example throughout the paper. The following section will introduce the formal model of the navigation system as well as the test case criteria and the test cases. In Section V we will present how the model is used to derive test cases automatically as well as assumptions made for the practical implementation. In the following section we will present an evaluation of the approach. In Section VII we will discuss some related research in the area of testing robot systems. Finally, we will conclude the paper and point out some future work.

II. The multi-agent navigation system

The multi-agent navigation system we are interested in is depicted in Figure 1. The system comprises a fleet of transport robots which transports boxes between different locations in a warehouse. In order to allow good scaling of the fleet's performance and high flexibility the robots are able to move freely in the environment. To coordinate their navigation, the fleet uses centralized and decentralized measures.

Fig. 1. Transport robots of the multi-agent system. © incubed IT

In order to ease the coordination of the robots, to respect restrictions imposed by the environment, and to allow a certain predictability of the behavior of the robots for their human co-workers traffic areas are used. These areas impose certain constraints on the movements a robot is allowed to perform in that area. For instance, such an area could be a one-way that forces the robot to move in a certain direction only. Additionally, some areas impose constraints how robots can interact with each other in these areas. For example, one can use a single robot area to ensure that only one robot is present in a certain area at any time. This allows for respecting given safety requirements or eases maneuvering to avoid for instance deadlocks at narrow passages.

In order to respect the limitations of the traffic areas and to perform a coordinated multi-agent navigation the system uses the navigation system presented in [6]. The implementation consists of a central server which coordinates the use of areas by individual robots and a decentralized planning algorithm which performs the planning on each robot using the information of the central server. The central server uses a set of reservations for each area in order to keep track when a robot plans to enter or leave a certain area. Furthermore, each robot needs to reserve an area before the robot is allowed to enter this area. This allows the central server to enforce proper cooperation between the robots with a minimal overhead. The planning system on the robots plan a path which is the fastest possible path respecting the geometrical structure of the environment as well as actual and future utilization of areas. Thus, the fastest path may not always be the geometrically shortest path but may also consider possible waiting times before a robot is allowed to enter a certain area or prefer a faster detours through alternative areas.

III. Running Example

In order to be able to show how we model the multi-agent navigation system we will use a simplified set of requirements derived from the original system as a running example. We will use these requirements in the subsequent sections to show how we model different aspects of the system.

We divide the requirements into two categories. The first category of requirements comprises requirements describing the traffic areas embedded into the environment. The second category represents requirements that describe the correct navigation of the robots within the environment.

We will use the following requirements for traffic areas:

1) The environment may contain several convex, non-overlapping traffic areas.
2) Each traffic area may have 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.
3) At any time it is not allowed that a robot is in a forbidden area.
4) A robot is only allowed to move into a predefined direction within a one-way area.
5) Two one-way areas are not allowed to be attached to each in a way that a robot which leaves a one-way area enters another one-way area.
6) At any time at most one robot is allowed within a single robot area.

Figure 2 depicts an example of an environment with traffic areas.

Beside the requirements for the traffic areas the requirements for correct navigation are the following:

1) A robot needs to find the fastest path within any environment for any given trajectory of other robots if one exists.
2) The fastest path is the path with the shortest execution time. Thus, a detour should be preferred if otherwise the robot needs to wait unnecessarily long to enter a certain area.

Although the requirements for the navigation task seems to be simple they already describe a quite complex task.

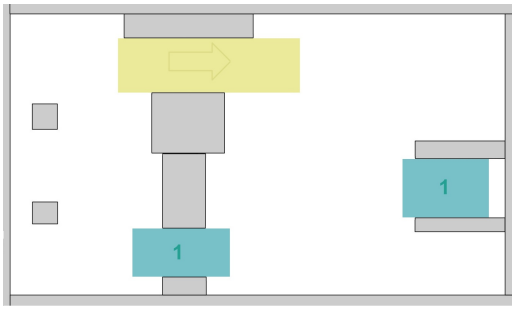


Fig. 2. Example environment with traffic areas. Green areas with a one inside represent a single robot area. One-ways are represented as yellow areas. Gray areas are forbidden areas.

In particular, the first requirement imposes a quite complex planning problem. The path which is used by the robot needs to be the fastest one, regardless the environment constellation. This includes different start and end areas as well as any area which may be used during the path. Additionally, the reservations which are made by any other robot needs to be considered in such a way that the global environment constraints are satisfied.

IV. Model-based test case generation

In order to be able to generate test cases for the multi-agent navigation system automatically we follow a model-based testing approach. The utilized model comprises two parts and is formed by a set of constraints. The first part is the domain model. It consists of a model of all possible and valid environments. In addition to the environment all possible traces through the environment are contained in the domain model. The second part is a declarative description of a particular property we want to test. We call this part the test case criteria. By solving the combined constraint system, we are able to derive test cases for the required property automatically.

A. Modeling the Multi-Agent Domain

The model of the domain comprises a constraint-based description of the potential layouts of the environment, the paths a single robot may take trough the environment, and the interaction of the robot with other robots.

1) Environment Model: In this work we are mainly interested in testing the proper handling of the traffic areas by the robot rather to test all details of the low-level navigation. Therefore, an abstracted topological representation of the environment is sufficient in the first place. We represent the traffic areas as a set of n areas $\mathcal{A} := \{A_1, A_2, ..., A_n\}$. The areas form the vertices of a colored directed graph $\mathcal{G} := (\mathcal{A}, \mathcal{E})$ where the set of edges $\mathcal{E}$ represents possible transitions between areas. The coloring function $type : \mathcal{A} \rightarrow 2^{\mathcal{T}}$ maps an area to a set of area types. This graph represents all possible environments that can be build up with n traffic areas allowing us to generate potential environments automatically. Please note that by assuming convex areas for traffic areas in the real environment not all environments can be realized in the real world. In section V we will discuss how we handle this issue.

Moreover, due to the coloring of the areas and given requirements further restrictions apply to valid environments. These restrictions are represented by a set of constraints $\mathbb{C}_{ENV}$. For instance for an area with the type *forbidden area* no incoming or outgoing edges are allowed: $\forall a \in \mathcal{A}.\mathcal{T}_{forbidden} \in type(a) \rightarrow \nexists a'.\langle a, a'\rangle \in \mathcal{E} \vee \langle a', a\rangle \in \mathcal{E}$. Moreover, a requirement exists that one-way areas must not be concatenated: $\nexists a, a'.(\langle a, a'\rangle \in \mathcal{E} \vee \langle a', a\rangle \in \mathcal{E}) \wedge \mathcal{T}_{one-way} \in type(a) \wedge \mathcal{T}_{oneway} \in type(a')$.

2) Path Model: In order to be able to test certain properties of the system we have to force the robot to traverse a path through the environment with particular properties. Such properties are for instance that the robot needs to traverse a transition between areas of a particular type. In order to define a path, we follow the standard definition of a path in a graph. A path is 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 $\forall i = 0...k-1, a_e^i = a_s^{i+1}$. The set of paths Π represent all potential ways a robot may take in the environment. In order to test particular properties, we need paths that start and end in particular areas. Therefore, we define a path $P_{a,b}$ that starts in area a ($a_s^0 = a$) and ends in area b ($a_e^k = b$). We denote the set of all paths between a and b as $\Pi_{a,b}$. Moreover, we are interested in the time a robot needs to traverse a path P. We define the travel time as the function $travel : \Pi \rightarrow \mathbb{R}$ with $travel(P) = \sum_{i=0...k-1} leave(a_s^i, a_e^i) + wait(a_s^i, a_e^i) + enter(a_s^i, a_e^i)$ where $leave(a_s^i, a_e^i)$ represents the time needed to leave area a_s^i towards area a_e^i along the path, $wait(a_s^i, a_e^i)$ represents the time a robot may wait before entering area a_s^i from area a_e^i and $enter(a_s^i, a_e^i)$ represents the time needed to enter area a_s^i from area a_e^i along the path. Finally, we need a definition of the shortest path $P_{a,b}^*$ between areas: $\forall p \in \Pi_{a,b}.travel(p) \geq travel(P_{a,b}^*)$. The constraints about paths are collected in the set $\mathbb{C}_{PATH}$.

3) Reservation Model: Areas are treated like resources. A robot needs to reserve an area in order to be allowed to traverse it. Depending on the type of the area, constraints apply to the possible reservations. If we assume that the other robots behave as expected the interaction between robots can be represented by these reservations. This allows us to represent the actual behavior of the multi agent system with the path of a robot and appropriate reservations only. A reservation is represented by the tuple $I := \langle a, t_s, t_e, r\rangle$ where a represents the reserved area, t_s respectively t_e the start time respectively end time, and r the reserving robot. We denote element a of tuple i as $i.a$. We collect all reservations in the set $\mathcal{I}$.

Using Allen's interval algebra [2] we are able to represent restrictions originally in the type of the areas and the actual path of the robot. For instance, all reservations of area of type 'single robot' need to be disjoint: $\forall i, j \in \mathcal{I}.i \neq j \wedge i.a = j.a \wedge \mathcal{T}_{oneway} \in type(i.a) \rightarrow i\{b, b'\}j$. A path of robot R is valid if all traversed areas are reserved by the robot. Please note in order to represent the path we need $k+1$ reservations. We express

this by the following constraint: $\forall_{i=0...k+1}\exists_{\langle a,t_s,t_e,r\rangle\in\mathcal{I}} a = a_s^i \wedge r = R \wedge \langle a_s^i, t_s^i, t_e^i, R\rangle\{s,f,d\}\langle a,t_s,t_e,r\rangle$. By assuming the path starts always at time $t = 0$ we define $t_s^0 = 0$, $t_s^1 = leave(a_s^0, a_e^0) + wait(a_s^0, a_e^0)$, and $t_s^{i+1} = t_s^i + enter(a_s^{i-1}, a_e^{i-1}) + leave(a_s^i, a_e^i) + wait(a_s^i, a_e^i)$ for $i = 2...k$. Moreover, we define $t_e^i = t_s^{i-1}$ for $i = 1...k-1$, and $t_e^k = t_e^{k-1} + enter(a_s^k, a_e^k)$. We collect these constraints in the set $\mathbb{C}_{RESERVE}$. Following this we name the shortest path that respects these additional constraints $\hat{P}_{a,b}^*$. Please note that this path may be longer than $P_{a,b}^*$ because the robot may be forced to wait in front of an area or may be forced to take a detour.

Let's consider a simple example to show how the intervals need to be reserved for a robot. The robot moves from one area (A_0) to another area (A_3) traversing through two different areas (A_1, A_2). The different times the robot spend to enter/leave the area as well as a waiting time before the robot can enter the area are depicted in Figure 3. Additionally, the intervals the robot needs to reserve the areas are depicted. Following the above conditions, the robot needs to reserve interval I_0 for area A_0 and so on. Thus the robot ensures that during the complete traversal of the environment the occupied space of the robot is reserved.

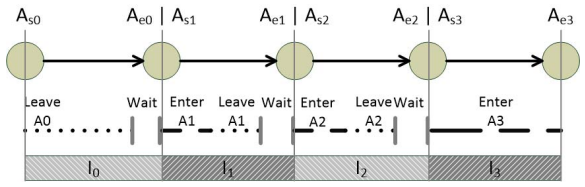


Fig. 3. Reserved intervals for traversing three areas

B. Test Case Criteria

If we combine the three constraint sets to a domain model $\mathbb{C}_{DOMAIN} := \mathbb{C}_{MAP} \cup \mathbb{C}_{PATH} \cup \mathbb{C}_{RESERVE}$ we get a model of all possible and valid environments and transfers within these environments. If we have a formal model of the robot's control software we could formally verify that the robot will behave according to the specifications [17], [12]. In this paper we follow the alternative approach of model-based testing (MBT) as no formal model of the controller exists. Using the above model, we are able to generate test cases for all different valid environments and navigation tasks. These test cases would represent the entire input space for the navigation system. Unfortunately, this space is huge if not infinite when no bounds are given for the environment. Therefore, it is not feasible or even not possible to execute all test cases.

Therefore, we focus the test case generation to specific aspects of the system. In particular, we want to test with a generated test case if a requirement given is implemented correctly. Therefore, we define a test case criterion that is derived from a requirement and restricts possible test cases to test cases related to the requirement. A test case criterion is basically a declarative description of a test case and is represented as the set of constraints $\mathbb{C}_{CRITERIA}$. This set is unified to the above model to restrict the possibilities to test cases related to the requirement. We can divide criteria into two classes. The first class is related to a problem in the path calculation. For instance, the shortest path is not found or the path traverses the environment in a forbidden way such as the wrong direction in a one-way area. The second class is related to problems in the management of reservations. For instance, the robot does not reserve all traversed areas properly, other reservations are ignored during planning or the planning does not deal correctly with its own reservations resulting in a path which includes unnecessary detours or waiting time. For instance, if we like to make sure that the robot under test is forced to take an optimal detour because areas along the optimal path are not available we represent the criterion as the following constraint: $\hat{P}_{a,b}^* \neq P_{a,b}^*$.

C. Test Cases

We define a test case for a single robot as the tuple $T := \langle \mathcal{G}, type, s, e, \mathcal{I}\rangle$. $\mathcal{G}$ is a topological graph and and $type$ is the coloring function for that graph. Together they represent the environment the robot acts in. s and e are areas that represent the origin and destination of the navigation task. $\mathcal{I}$ is a set of reservations representing the reservations and therefore restrictions imposed by other robots. This information is sufficient to execute the test case on the robot system by generating the encoded environment and commanding the restricted navigation task. Test cases for a particular test case criterion can be obtained by solving the combined constraint system $\mathbb{C} = \mathbb{C}_{DOMAIN} \cup \mathbb{C}_{CRITERIA}$. Moreover, because $\mathbb{C}$ describes all valid paths in respect to domain description and test case criteria it can also be used as test oracle to validate the test case execution. If invalid reservations are made by the robot or the robot follows an invalid path an error in the implemented system had been found.

V. Test case generation

In the previous section we have defined a test case T as well as the model of potential environments and tasks and a test case criterion that can be used for test case generation. In theory a test case could now be derived by combining all the constraints into one set $\mathbb{C}$ and solving the constraint system for an assignments of the elements presenting the test case T. Unfortunately, this direct approach is not feasible in practice. The reason for this is the structure and complexity of certain constraints.

First, the environment represented as a graph of areas and area transitions needs to be drawn on a two dimensional plane using non-overlapping convex areas. In order to ensure that a graph can be drawn the graph needs to be at least planar [18]. Due to the restrictions to convex areas additional constraints needs to be applied [4]. Due to these constraints generating environments using a standard constraint solver is infeasible for a significant large number of areas.

The second problem of the direct approach is the definition of a path. A path is a sequence of connected area transitions.

As such the definition itself needs a higher order specification. If no limit is given for the length of the sequence one needs to consider a recursive definition which can deal with infinite sequences. Similar to the previous case a standard constraint solver cannot process this information appropriately.

Finally, the set of intervals representing reservations causes difficulties. As the size of the set needed to enforce a test case criterion is not known a prior, one needs to consider definitions which can deal with potentially infinite sets. Due to the needed definition a standard constraint solver cannot deal with such intervals efficiently.

In order to address these problems and to be able to use a standard constraint solver we combine a divide-and-conquer approach with iterative deepening and a roll-out of constraints. First we will split the generation process into three parts and will address each part separately. Algorithm 1 shows how the splitting of the problem is performed. The algorithm uses a test case criteria and a set of graph templates to generate all elements of a test case.

Algorithm 1: Generation of test cases for a multi-agent navigation system

```
   Data: C_ENV ... the environment constraints
   Data: C_PATH ... the path constraints
   Data: C_INTERVAL ... the interval constraints
   Data: C_CRITERIA ... a test case criterion to fulfill
   Data: Γ ... set of graph templates
   Output: G ... the topological graph of the environment
   Output: T ... an assignment of types for each area
   Output: s ... the start area
   Output: e ... the end area
   Output: I ... a set of reserved intervals
1  begin
2      foreach ⟨G_TEMP, C_TEMP⟩ ∈ Γ do
3          while ⟨G, T⟩ ← genEnv(G_TEMP, C_TEMP, C_ENV) do
4              while ⟨s, e⟩ ← genPath(G, T, C_PATH, C_CRITERIA)
                   do
5                  if I ←
                       genInt(G, T, s, e, C_PATH, C_RESERVE, C_CRITERIA)
                       then
6                      return ⟨G, T, s, e, I⟩
7                  end
8              end
9          end
10     end
11     return nil
12 end
```

The algorithm iterates through the given graph templates. In order to be able to realize the graph as a physical environment a graph template is based on a grid of a particular size $n \times m$. Each grid cell represents a possible area and vertex in the graph. For practical reasons we currently limit areas in the physical environment to rectangles. Please note that not all cells must be used allowing different arrangements of areas like crosses which are well suited for testing the behavior at crossings. A graph template is represented by a set of binary variables $\mathbb{G}_{TEMP}$ representing the adjacency matrix and a set of constraints $\mathbb{C}_{TEMP}$ representing restrictions in the connectivity in the template. This information is used to generate a concrete environment.

If an environment can be generated, the algorithm uses the environment to generate a start and an end area of the navigation task which respects the test case criteria. Finally, a set of intervals is generated that enforces the test case criteria. If one of the inner test case generation steps fail, the outer iteration ensures that another valid instantiation will be generated. By using this simple iterative procedure, the complete space of possible environments, paths and intervals will be searched to find a test case that respects all constraints. It is important to notice that most of the time only a small number of iterations is necessary to generate a test case.

Algorithm 2: Generation of a valid environment

```
   Data: G ... adjacency matrix of graph template
   Data: C_TEMP ... constraints on graph template
   Data: C_ENV ... environment constraints
   Output: G ... complete instantiation of G
   Output: T ... complete type assignment of areas in G
1  begin
2      T = {T_1, ...., T_k}, k = #areas in G, dom(T_i) = T
3      if solveConstraints(G ∪ T, C_TEMP ∪ C_ENV) then
4          return ⟨G, T⟩
5      end
6      return nil
7  end
```

Algorithm 2 uses a graph template to generate an environment representation. It first creates variables to represent the type assignment of the different areas. The variables of the template and the type assignment together with the constraints on the template and the environment form a constraint satisfaction problem. By using a constraint solver, a valid variable assignment is obtained. If an assignment is found the graph and the type assignment is returned. Please note that a modern solver allow to iterate through alternative solutions. Therefore, in order to report alternative solutions to the iteration used in Algorithm 1 we report alternative solutions instead of solving the problem every time excluding previous solutions.

The generation of a start and end area of a navigation order which imposes an optimal path that satisfies the test case criteria needs to address the second problem defined at the beginning of this section. The problem is the definition of a path which needs to treat a possible infinite path properly. This can be addressed if we fix the length of a potential path to a certain size k. For a fixed size the sequence of transitions can be simply represented by an array of area variables of size $2 \times k$. In order to find the right length an iterative deepening approach can be used. The approach iterates trough increasing path length from 1 till $|\mathcal{A}|$. Thus we allow paths that traverse all areas only once. Moreover, we are interested in the shortest valid path. Finding this path is guaranteed by reporting the first path found by the iterative deepening approach. Thus cycles need not to be represented.

Algorithm 3 performs such an iterative deepening approach to generate a start and end area for the test case. It iterates trough increasing path lengths. In each iteration the algorithm creates variables for transitions representing a path with a given length k. $P_{2\times i-1}$ and $P_{2\times i}$ represent the i^{th} transition

Algorithm 3: Generation of a start and end area satisfying the test case criteria

Data: $\mathbb{G}\ldots$ instantiated adjacency matrix of graph
Data: $\mathbb{T}\ldots$ an assignment of types for each area in $\mathbb{G}$
Data: $\mathbb{C}_{PATH}\ldots$ the path constraints
Data: $\mathbb{C}_{CRITERIA}\ldots$ a test case criterion to fulfill
Output: $s\ldots$ the start area of the path
Output: $e\ldots$ the end area of the path

```
begin
    for k ← 1 to |𝔸| do
        ℙ = {P_1, ..., P_{2×k}},  dom(P_i) = 𝒜
        if solveConstraints(𝔾 ∪ 𝕋 ∪ ℙ, ℂ_PATH ∪ ℂ_CRITERIA)
          then
            s = P_1
            e = P_{2×k}
            return ⟨s, e⟩
        end
    end
    return nil
end
```

of the path. These variables together with the variables of the environment are combined with the constraints for paths and the test case criteria to form the related constraint satisfaction problem. Then the algorithm solves the constraint problem to find a variable assignment representing a valid path. If such an assignment exists, the variables for the start and end area of the path are extracted and returned.

In order to generate a test case, the last step is the generation of a set of intervals representing reservations of areas by other agents. In order to generate such a set one needs to address the unknown size of the set of necessary intervals imposed by the test case criterion. In order to address this problem, we will use again an iterative approach with an upper bound. As intervals are only of interest for the path or any alternative path between start and goal we can derive an upper bound on the number of intervals as $k \times |\Pi_{s,e}| \times 3$, where k is the length of the longest path in $\Pi_{s,e}$. Due to the restriction of the intervals to the path or its alternatives we need at most one interval for each transition on the potential paths. Additionally, an interval can be either before, after or during such a transition.

Algorithm 4 performs such an iterative approach to generate a set of intervals respecting the test case criterion. First the algorithm calculates the longest path between s and e. This calculation can be simply performed as a graph search in $\mathbb{G}$. Then the algorithm iterates through increasing sizes for the set of potential intervals. In each iteration variables for a potential alternative path and the intervals are generated. Additionally, constraints to fix the start and the end area of the path according to the input are generated. All variables and constraints are combined to a constraint problem and a constraint solver is used to find a possible variable assignment. If such an assignment is found the set of intervals is extracted and returned. Please note that the upper bound for the number of iterations is valid for test case criteria dealing with paths without cycles. It may not be a valid upper bound for other criteria.

Before we present the evaluation of the approach based on the industrial use case we will briefly walk through a simple

Algorithm 4: Generation of an interval set satisfying the test case criteria

Data: $\mathbb{G}\ldots$ instantiated adjacency matrix of graph
Data: $\mathbb{T}\ldots$ an assignment of types for each area in $\mathbb{G}$
Data: $s\ldots$ the start area of the path
Data: $e\ldots$ the end area of the path
Data: $\mathbb{C}_{PATH}\ldots$ the path constraints
Data: $\mathbb{C}_{RESERVE}\ldots$ the reservation constraints
Data: $\mathbb{C}_{CRITERIA}\ldots$ a test case criterion to fulfill
Output: $\mathcal{I}\ldots$ the set of intervals

```
begin
    k = longestPathBetween(𝔾, s, e)
    for i ← 0 to k × |Π_{s,e}| × 3 do
        ℙ = {P_1, ..., P_{2×k}},  dom(P_i) = 𝒜
        𝕀^a = {I^a_1, ..., I^a_k},  dom(I^a_i) = 𝒜
        𝕀^{t_s} = {I^{t_s}_1, ..., I^{t_s}_k},  dom(I^{t_s}_i) = ℕ
        𝕀^{t_e} = {I^{t_e}_1, ..., I^{t_e}_k},  dom(I^{t_e}_i) = ℕ
        𝕀^r = {I^r_1, ..., I^r_k},  dom(I^r_i) = ℛ \ R
        ℂ_P = {P_1 = s, P_{2×k} = e}
        𝕍 = 𝔾 ∪ 𝕋 ∪ ℙ ∪ 𝕀^a ∪ 𝕀^{t_s} ∪ 𝕀^{t_e} ∪ 𝕀^r
        ℂ = ℂ_P ∪ ℂ_PATH ∪ ℂ_RESERVE ∪ ℂ_CRITERIA
        if solveConstraints(𝕍, ℂ) then
            return {⟨a^i, t^i_s, t^i_e, r^i⟩|
              a^i = I^a_i ∧ I^a_i ∈ 𝕀^a, t^i_s = I^{t_s}_i ∧ I^{t_s}_i ∈ 𝕀^{t_s}, t^i_e =
              I^{t_e}_i ∧ I^{t_e}_i ∈ 𝕀^{t_e}, r^i = I^r_i ∧ I^r_i ∈ 𝕀^r}
        end
    end
    return nil
end
```

example. We like to generate a test case respecting the test case criteria that the robot needs to traverse an area of type 'single robot' on the shortest path but a detour is faster than the direct navigation.

The first step to generate a test case is the utilization of graph templates. After some iterations the algorithm uses the graph template depicted in Figure 4a. The graph template defines the seed for an environment where all possible connections are allowed within the environment.

By using the graph template, the algorithm generates the fully specified environment depicted in Figure 4b. Please note that the environment contains two areas $A1$ and $A4$ of type 'single robot'. Additionally, the environment defines all connections between the areas.

Using the environment and the test case criteria the algorithm generates start and end areas for the test case. The used start and end areas are depicted in Figure 4c. Please note that the shortest path leads through area $A1$ which is an area of type 'single robot'.

As we are interested in a test case where a detour should be preferred over the shortest path the algorithm needs to generate the reservations accordingly. The shortest path and a valid detour is depicted in Figure 4d. As discussed above the shortest path leads through $A1$. If the area is reserved for another robot (r') the area cannot be traversed by the robot. Instead the robot would need to wait until robot r' leaves the area $A1$. Thus the algorithm needs to generate a reservation for area $A1$ which causes a waiting time longer then the costs of the detour. Let's consider the time from start to end area if the detour is taken is 50 seconds. The algorithm generates the

following reservations $\mathcal{I} = \{\langle A1, 0, 51, r' \rangle\}$ based on the test case criterion. Thus the area $A1$ is reserved longer than the time needed for the detour ensuring that the detour is faster than the direct way. This yields to test case depicted in Figure 4d.

VI. Evaluation

In order to evaluate the proposed model-based testing approach empirically we implemented the test case generation using the programming language Java and the CSP solver library Choco [14]. The test case generation for the experiments ran on an i5 with 8GB of RAM running Ubuntu 14.04 and Java 1.7.

In order to evaluate that the automatically generated test cases are suitable to find potential errors in the navigation system we executed generated test cases using the actual industrial implementation of the navigation system described in [6]. For the evaluation a multi-robot simulation environment based on Stage was used that replicates the environment and the movement of the robots within the environment. This setup was already used for manual testing. The generated test cases were converted into descriptions of the environment including traffic areas, the navigation task for the robot that executes the test, and area reservations that represent movements of other robots that can be loaded by the central server and the decentralized planner of the robot under test.

For the evaluation we generated a test suite for the complete navigation system using the method described in the previous section. For the generation we used the different test case criteria for path planning shown in Table I and for reservation handling shown in Table II. The test suite comprises 14 test cases which were generated in 1.6 seconds on average. This clearly shows that through the splitting of the generation process in several steps the time to produce a test suite is reasonable.

test case criteria
the path moves only within an area with a certain type
the path needs to start in an area with a certain type
the path needs to end in an area with a certain type
the path needs to lead through an area with a certain type
the path needs to have a transition of two areas with a certain type combination
only one path is possible between two areas

TABLE I
Test case criteria related to wrong behavior in planning a path.

test case criteria
reservation before the robot needs to traverse an area
reservation after the robot needs to traverse an area
reservation on all areas for the complete time except the path during its necessary traverse time
reservation of alternative paths with same traversal time
enforcing that a detour is faster than the shortest path
enforcing that a detour is not faster than the shortest path

TABLE II
Test case criteria related to wrong behavior in reserving areas.

Using this test suite 68.14 % of lines of code of the complete navigation system were covered. Moreover, 31.07 % of the branches of the complete navigation system were tested. It is important to note that some lines may be only covered in the case a re-planning on the reactive level needs to be performed by the robot. As we have not specified the behavior for re-planning at that level no test cases covering these parts of the system were generated. Finally, it is important to mention that many of the branches are used to cover cases of invalid input or invalid interaction of software modules. As we did not generate test cases for invalid input or invalid interaction these branches were not covered neither.

In order to check how efficiently a potential flaw can be found we created mutated versions of the implementation of the navigation system. Mutants were created manually by mutating conditions in the code. Such mutations follow the idea of mutation-based testing where mutants represent usual mistakes made by programmers [5]. We created 92 mutated versions, where 98.92 % of them were detected by using the automated generated test suite. This shows that the generated test suite is able to catch a large fraction of possible implementation flaws.

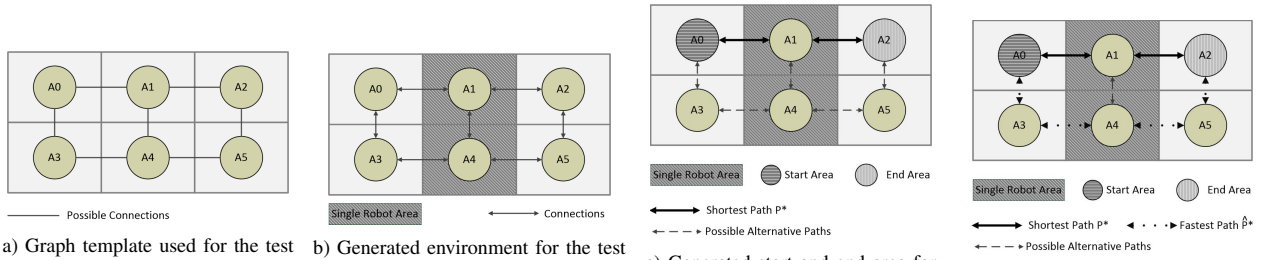
It is important to note that one actual flaw in the live implementation of the system was detected causing the system to crash with an illegal memory access. While manual testing was not able to revile that flaw the generated test cases reviled this issue.

Summarizing, one can state that the generated test cases were able to cover a large fraction of possible implementation flaws. Moreover, they were able to detect actual flaws in the implementation which stayed undetected before using manual testing. Thus the evaluation showed that the presented methods are applicable and suitable for testing complex multi-robot navigation system.

VII. Related Research

The work presented in [9] used a model-based approach to generate test cases for testing the interaction of a robot and its environment. The model was a simple label transition system which uses states to represent the environment and robot state and labels to represent events which were issued by the robot or observations made in the environment. Additionally, internal feedback of the robot system was described by separate labels. By using a random walk through this model test cases were generated. This is in contrast to our approach which performs a more structured generation of test cases in order to ensure certain coverage criteria.

To test the robustness of the functional layer of a robot against invalid requests a test case generation schema was discussed in [13]. The method uses a seed input which is expected to represent correct requests. This golden model is then mutated to create invalid requests that need to be handled properly by the system. In order to check if the system reacts properly the reaction of the functional layer was observed and checked against a formal description of requested properties. Thus the generated tests can be seen as a kind of random



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

Fig. 4. Test case generation

tests where in contrast our approach explores the input space in such a way that a certain coverage of the requirements is achieved.

In [1] the authors argue that to test an autonomous robot properly one needs to cover the situation the robot could be in. This should be achieved by defining coverage criteria for the situation as well. This is actually the approach we follow in this work. We have defined an abstracted formal description of possible situation. Although we do not explore the space of possible situations directly we search for situations that cover certain requirements. Additionally, our approach allows to generate test cases for given criteria automatically. Currently the used criteria are more focused towards covering requirements but instead it could be related to situation directly in the sense of testing a potential input space.

The work presented in [15] creates test cases for a robot navigation system according to different parameters that resemble the difficulty of the navigation task. In order to generate different test cases, the parameters are varied to generate environment with different terrain and obstacles and navigation tasks. The generated tests can be classified as difficult or easy. The authors showed that there is a correlation between these parameters and the difficulty. In contrast to our approach a broad set of test cases was generated and classified after execution based on the outcome. The aim is to find test cases that have a certain classification.

In order to generate tests automatically the use of an ontology was proposed in [11]. The ontology describes the interactions that are allowed between different agents. By using a domain ontology and an ontology alignment valid and invalid input data are generated for testing. For inputs where no specification is given in the ontology, rules are used to generate valid input. To explore the input space those inputs are preferred to be issued that were not used before. This is in contrast to our approach that selects the input to explore the space of possible behaviors of the system rather than enforcing a desired behavior directly.

A more focused method was presented in [10]. The authors propose to use an evolutionary algorithm to drive the inputs towards test cases that maximize the chance to detect some flaw. The idea is to specify an objective function which measures how good a certain requirement is implemented within the agent. During the test case execution this objective function is measured and used to steer the test case generation towards those constellations that have a higher chance that the requirement is violated. New test cases are generated by mutating old ones. This is in contrast to our approach which tries to generate a test case that covers a certain aspect of the requirement. Furthermore, our approach allows to create valid test cases even for situations where it would be hard that a randomly mutated test case is valid which would be the case if an evolutionary algorithm is used.

In [8] the authors propose to generate test cases using a model of the context of the robot and a description of the requirements. The context model uses an ontology to describe the structure of the environment in an abstract way. The requirements are modeled using UML sequence diagrams. To generate test cases a search-based approach is used that is based on automated modification of contexts. The approach used a fitness function which tries to minimize the number of context models, to maximize the number of requirements covered and lead to the boundary of an allowed domain. After the execution of the generated test cases different coverage criteria can be applied to check context or requirements coverage. This is also the main contrast to our approach as we try to find those test cases which yield a specific coverage of contexts and the requirements.

VIII. Conclusion and Future Work

Intelligent autonomous multi-robot systems get increasingly deployed in industrial settings like warehouse automation. Due to the close interaction with humans and economical needs dependability of such systems is of particular interest. Due to the complexity and richness of the behavior of such systems and their environment proper manual testing is tedious. Thus in this paper we showed how to automate the testing. This is done through model-based testing which was applied for an industrial use case where a fleet of autonomous robot's transport goods in a warehouse.

In this paper we focused on the testing of the cooperative navigation system used by the robots that incorporates operational and structural limitations of the environment. We presented an approach to obtain a formal model based on

constraints that represent all potential situations comprising an environment and the behavior of the robot fleet. By reusing properties of the multi-agent navigation system we were able to focus that model towards a single robot view. Using this model, we were able to generate expressive test cases for every potential situation. Because executing all these test cases is impossible we focused the test case generation towards test cases that cover particular requirements. A direct generation is computationally infeasible due to the usage of higher order constraints. We presented a fast, practical implementation of the test case generation that is based on a divide-and-conquer approach which is combined with iterative deepening. This allowed us to generate test cases quickly using a standard constraint solver. We evaluated the proposed test case generation using a real industrial use case. The evaluation showed that we were able to generate suitable test cases quickly. Furthermore, these test cases were able to cover large parts of the system's code and were able to uncover most errors artificially introduced in the code.

So far we have only used a predefined set of coverage criteria to steer the test case creation. Thus it is left for future work to use other methods to better explore the model of the system for test case generation. One such method could be model-based mutation testing [7] which aims to detect faults using deliberatively altered versions of the model for test case generation. Moreover, we have only tested the navigation system according to the proper use of traffic areas. It would be of interest to expand the test case generation also towards unknown or moving objects in the environment allowing to test the low-level navigation. Finally, to address the potentially infinite number of situations it might be interesting to find equivalence classes within the modeled environments and tasks.

REFERENCES

[1] R. Alexander, H. R. Hawkins, and A. J. Rae. Situation coverage–a coverage criterion for testing autonomous robots. Technical Report YCS-2015-496, Department of Computer Science, University of York, 2015.

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

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

[4] 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.

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

[6] 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.

[7] W. Krenn, R. Schlick, S. Tiran, B. Aichernig, E. Jobstl, and H. Brandl. Momut:: Uml model-based mutation testing for uml. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–8. IEEE, 2015.

[8] Z. Micskei, Z. Szatmári, J. Oláh, and I. Majzik. A Concept for Testing Robustness and Safety of the Context-Aware Behaviour of Autonomous Systems. In *Agent and Multi-Agent Systems. Technologies and Applications: 6th KES International Conference, KES-AMSTA*, pages 504–513, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[9] C. Mühlbacher, S. Gspandl, M. Reip, and G. Steinbauer. Improving dependability of industrial transport robots using model-based techniques. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3133–3140. IEEE, 2016.

[10] C. D. Nguyen, S. Miles, A. Perini, P. Tonella, M. Harman, and M. Luck. Evolutionary testing of autonomous software agents. *Autonomous Agents and Multi-Agent Systems*, 25(2):260–283, 2012.

[11] C. D. Nguyen, A. Perini, and P. Tonella. Ontology-based test generation for multiagent systems. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1315–1320. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[12] M. O'Brien, R. C. Arkin, D. Harrington, D. Lyons, and S. Jiang. Automatic verification of autonomous robot missions. In *Simulation, Modeling, and Programming for Autonomous Robots: 4th International Conference, SIMPAR 2014*, pages 462–473. Springer International Publishing, 2014.

[13] D. Powell, J. Arlat, H. N. Chu, F. Ingrand, and M.-O. Killijian. Testing the input timing robustness of real-time control software for autonomous systems. In *Dependable Computing Conference (EDCC), 2012 Ninth European*, pages 73–83, 2012.

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

[15] T. Sotiropoulos, J. Guiochet, F. Ingrand, and H. Waeselynck. Virtual worlds for testing robot navigation: a study on the difficulty level. In *12th European Dependable Computing Conference (EDCC 2016)*, 2016.

[16] 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.

[17] H. Täubig, U. Frese, C. Hertzberg, C. Lüth, S. Mohr, E. Vorobev, and D. Walter. Guaranteeing functional safety: design for provability and computer-aided verification. *Autonomous Robots*, 32(3):303–331, 2012.

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

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

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