# Multi-Robot Surveillance Task Planning Under Uncertainty

António Matos
antonio.a.matos@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2021

**Abstract**

The usage of multi-robot teams has increasingly been considered as a good solution for improving reliability and decreasing down-times in industrial and scientific applications. With the advent of the multi-robot team, challenges have emerged in coming up with efficient coordination strategies that take into account uncertainty in regards to action duration. Generalized stochastic Petri nets with rewards (GSPNRs) models that interpret tokens as robots provide a formalism capable of modeling these multi-robot scenarios with a compact state space, whilst also allowing an explicit stochastic modeling of action execution time. The present thesis extends recent approaches based on these GSPNRs in order to model heterogeneous teams made up of multiple types of robots and to allow the inclusion of a single robot-specific attribute such as battery. Additionally, a software toolbox for MATLAB is developed that provides an implementation of GSPNRs and methods to calculate optimal policies on these models. The software package also includes the possibility of executing these policies on real robots by interfacing with Robotic Operating System (ROS) middleware. Finally, an inspection task planning scenario is solved and results were obtained in realistically simulated robots in Gazebo.

**Keywords:** Multi-Robot Systems, Multi-Robot Coordination, Planning Under Uncertainty, Generalized Stochastic Petri Nets with Rewards, Heterogeneous Multi-Robot Systems

## 1. Introduction

Small, autonomous robots have become widely used in both in scientific and industrial fields. While these small robots have become more economical and easier to deploy they generally suffer from a significant shortfall: they are solely battery powered, and limitations present on current battery technology account for agents with relative short autonomy. On the other hand, their low-cost allows for the deployment of multiple robots at once, and recent research has examined the use of multi-robot teams to mitigate these battery limitations. The utilization of heterogeneous teams consists in using together different robot types with distinct capabilities. The main advantage of heterogeneous teams is to allow for solutions that exploit each robot-types strengths more effectively.

Multi-robot heterogeneous teams only provide such advantages when correctly and efficiently coordinated. Moreover, the use of heterogeneous teams originates the problem of enforcing cooperation while maintaining efficiency. A promising line of research advocates the use of formal models to capture the multi-robot problem and the use of dynamic programming algorithms to solve these decision-making and planning problems. These approaches provide formal guarantees such as showing that the multi-robot system has no deadlocks, and at the same time allow synthesizing optimal policies that coordinate the multi-robot system. Nonetheless, there is a lack modelling approaches that can capture heterogeneous teams in a compact manner, while also being able to track robot-specific attributes such as battery and uncertainty in the duration of action execution.

### 1.1. Objective and Contributions

The main objective of this thesis is to develop methods that can coordinate heterogeneous multi-robot systems by optimizing some defined performance criteria, while taking into account various constraints. Even though approaches that we developed are generic and can be applied to wide range of problems in diverse application areas, we chose to solve a specific multi-robot problem of interest to the research group:

**Inspection Problem** - Consider an inspection scenario, where the photo-voltaic panels in different locations of a solar farm must be continuously inspected by several small unmanned ground vehicles (UGVs) with limited autonomy, and a single larger UGV is used to recharge the smaller ones and so increase the autonomy of the system. **How should the robot team be coordinated in order to make this inspection process as efficient as**

**possible? How can we take into account the battery constraints of the small UGVs and the uncertainty associated with the navigation, inspection and charging actions?**

To solve this multi-robot planning problem, this research will use Generalized stochastic Petri nets **with rewards** (GSPNRs). This formalism will be used to model the inspection task planning problem, and then convert it into the equivalent Markov Decision Process (MDP) model, where policy synthesis algorithms can be applied, that extract the optimal policy by optimizing the utility function while taking into account uncertainty in the duration of actions. This problem has been solved as reported in [1] and [2]. This work introduces three novel contributions, required to solve the motivating problem:

1. Extend current formal models based in GSPNRs into a general framework that is capable of modelling a heterogeneous multi-robot task planning problem. It will also include the ability to model system wide resources, e.g. counters, or robot specific attributes, e.g. battery.

2. Build and test a software package in MATLAB environment capable of building GSPNR models, obtaining optimal policies for these models, and executing the obtained policies in real robots, by integrating with the Robotic Operating System (ROS).

3. Solve the specific inspection task planning problem. This will use the modelling framework established, and utilize the software package developed to obtain and execute optimal policies in simulated robots and environment.

## 2. Preliminaries - GSPNRs

A generalized stochastic Petri net with rewards is a tuple $G_R = (P, T, I(.), O(.), W(.), \boldsymbol{m}_0, r_P, r_T)$. $P$ is a non-empty, finite set of places. $T$ is a non-empty, finite set of transitions, that is partitioned into two subsets: $T = T_I \cup T_E$. $T_I$ is the set of immediate transitions, and $T_E$ the set of exponential transitions. Immediate transitions fire instantaneously, and exponential transitions fire with a stochastic delay determined by an exponentially distributed random variable. $I(.) : P \times T \to \mathbb{N}$ is a function defined over the set of input arcs (i.e. arcs connecting places to transitions) specifying their multiplicity. The multiplicity of each input arc corresponds to the number of tokens consumed in the place connected to the transition that fires. $O(.) : T \times P \to \mathbb{N}$ is the output multiplicity function, defined over the set of output arcs (i.e. arcs connecting transitions to places). For output arcs, the multiplicity corresponds to the number of tokens created in the place connected to the transition that fires. $W(.) : T \to \mathbb{R}_{\geq 0}$ is a weight function that maps a transition $t_k$ to a real positive number, and is denoted by $W(t_k)$. This value $w_k$ denotes the firing rate that characterizes the exponential distribution of exponential transitions, or the weight of an immediate transition. $\boldsymbol{m}_0 = (m_{01}, m_{02}, \cdots, m_{0P})$ is the initial marking. A marking $m$ is an assignment of tokens to places, and is represented by a vector, where each element corresponds to the number of tokens assigned to place: $\boldsymbol{m} : P \to \mathbb{N}$. Place rewards are a mapping between places and real numbers, $r_P : P \to \mathbb{R}$. They represent reward accumulated for each time unit that the place is marked by at least one token. A marking reward function is defined, mapping between markings and rewards $R : M \to \mathbb{R}$. This function maps each reachable marking $\boldsymbol{m} \in R(G)$ to a real number which is the sum of all place rewards in a given marking. Transition rewards represent a reward accumulated when a given transition fires, and so is a mapping between transitions and real numbers: $r_T : T \to \mathbb{R}$

## 3. GSPNR Modelling

### 3.1. Homogeneous Systems

In multi-robot homogeneous systems where all robots have the same capabilities, GSPNRs models can provide a compact way of representing the overall team state. Robots are represented as tokens, and each place in the GSPNR represents a particular state in which a robot may be at. If such a place is marked by a token, this means that a robot is currently in that particular state. The overall team state is represented by the marking of the GSPNR, specifying all the states in which each robot composing the team is at.

Transition firing is the mechanism with which tokens transverse through the places in a GSPNR. In a multi-robot problem where tokens represent robots, GSPNR transitions model any event in which the robot represented by a token changes state. The place in which the token was previously at represents the local state the robot was before the event took place. The place where the token is created by the transition firing models the state in which the robot lands after the event takes place.

Immediate and exponential transitions model different types of events. An instantaneous event such as a decision is modelled with an immediate transition. Thus, action selection and its non-determinism is captured with several immediate transitions connected to the same place by an input arc. When a token reaches a place connected to several immediate transitions, every conflicting immediate transition represents a different decision that the robot can choose to take. Exponential transitions represent an uncontrollable timed event and so are chiefly used to model uncertainty regarding action execution. When a token reaches a place

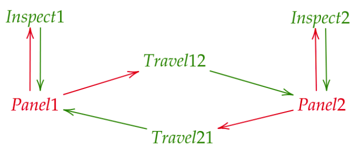


Figure 1: Decision-action graph for an inspection scenario; in green are decision nodes, and in red are action nodes;

connected to an exponential transition by an input arc, the time spent by the token in that place is a random variable exponentially distributed.

## 3.2. Decision-Action Graphs

In most multi-robot problems involving sequential decision-making, the states that a robot can occupy belong into one of two categories. A robot can be in a state where it must decide between executing several available actions, or it can be in a state of executing an action. After finishing action execution, the robot transitions again into a state where it must make another decision to execute another action. This process occurs repeatedly for all robots in the system. GSPNR models that capture this kind of "Decision/Action/Decision/..." problem contain an underlying structure. Formally defining this underlying structure provides a more direct interpretation between the multi-robot problem and its GSPNR model. Moreover, by defining this concept, algorithms to build GSPNR models are easier to understand and implement.

This structure in GSPNR models is described using decision-action graphs (*DecAct-G*). Formally, a decision-action graph is a bipartite directed graph $\mathcal{DA} = (D, A, E)$, where $D$ is the decision nodes set, $A$ is the action nodes set, and $E$ is the edges set. Each decision node represents a state where the robot must choose between various possible actions, and each action node represents an action that takes non-zero time to execute. Edges that begin in a decision node and end in an action node are denominated *decision edges* and represent all the possible actions that the robot can execute in that decision node; edges that begin in an action node and end in a decision node are denominated *outcome edges* and represent the decision state that the robot ends up in when the action has finished executing. The number of decision edges of an action node must be the same as the number of outcome edges, as the number of robots remains constant before and after executing an action.

Figure 1 depicts the *DecAct-G* that represents a homogeneous multi-robot problem where robots are able to carry out an inspection action in two separate locations and they are also able to travel between the two locations.

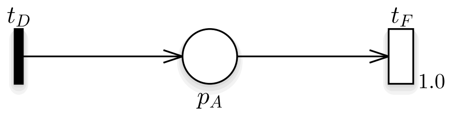


Figure 2: An action model for arbitrary action $A$; the immediate transition $t_D$ models the decision to execute $A$, and the exponential transition $t_F$ models finishing the execution of $A$;

Action nodes directly represent states where the robot is executing an action. Each action state is associated with two events: 1) the controllable event of deciding to start action execution; 2) the uncontrollable event of finishing executing the action. This leads us to define a function over the action nodes $ActionModel : A \to G$, where $A$ is the set of all action nodes and $G$ the set of all possible GSPNRs. $ActionModel(a)$ is the GSPNR that models the execution of action $a$. $ActionModel()$s have only three elements:

- *Decision transition* - an immediate decision representing the controllable event of deciding to execute the action;
- *Action place* - the place that models the robot state of actually executing the action;
- *Final transition* - an exponential transition modelling the uncontrollable event of finishing executing the action;

An action model for an arbitrary action $a$ is represented in Figure 2 and is formally defined as a GSPN $G_A = (P, T, I(.), O(.), W(.), \boldsymbol{m}_0)$ with the following restrictions $P = \{p_A\}$; $T = \{t_D, t_F\}$; $I(p_A, t_F) = 1$; $O(t_D, p_A) = 1$; $\boldsymbol{m}_0 = (0)$.

*DecAct-G*s allows us to build a multi-robot GSPNR model in a precise manner. Each node and arc in the graph is substituted by a corresponding GSPNR element, and they are merged together to create an overall GSPNR model that captures the multi-robot problem considered. To build the full GSPNR model from a $\mathcal{DA} = (D, A, E)$, each decision node is interpreted as a place denominated *decision place*, and each action node $a$ is replaced by its GSPNR model $ActionModel(a)$. Any decision edge $(d, a)$ is substituted by an input arc from the corresponding decision place to the decision transition $t_D \in ActionModel(a)$ of the action model corresponding to the action node. Any outcome edge $(a, d)$ is replaced by an output arc from the final transition $t_F \in ActionModel(a)$ to the decision place corresponding to the decision node.

### 3.3. Extension to Heterogeneous Systems

Until now, all tokens represented robots that were the same type, with the same capabilities. This means that robots could be represented anonymously as tokens in the GSPNR model. To extend this framework to model a heterogeneous robot team, the GSPNR model must unambiguously determine which type of robot each token represents. This is done by partitioning the GSPNR place set into several mutually exclusive subsets. Each subset is associated with a single robot type. By doing so, each token can be directly mapped to a specific robot type by examining which subset the marked place belongs to. Suppose we have a GSPNR $G = (P, T, I, O, W, r_P, r_T)$, and the multi-robot problem involves $n$ different robot types $R = \{r_1, \cdots, r_n\}$. The place set $P$ is partitioned into $n$ subsets: $P = \bigcup_n P_i$ which are all mutually exclusive $\forall i, j : P_i \cap P_j = \emptyset$. Tokens in a place belonging to the subset $P_i$ represent robots of type $r_i$.

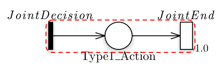
For homogeneous systems, the GSPNR model had to be conservative: the number of tokens needed to remain constant for all possible markings. This translates the natural constraint of keeping the number of robots constant, as robots cannot be created or destroyed. When extending the framework to heterogeneous systems, this constraint needs to be stronger. The number of tokens in each place subset representing different types of robots must remain constant. In practical terms, this forbids transitions connecting places belonging to different robot type subsets. Such a transition would imply transforming a robot of a particular type into another robot type.

Decision-Action graphs can also be used to build heterogeneous GSPNR models. By partitioning the decision node set into separate sets that are each associated with a single robot type, the mapping between places and robot types defined previously can be established when building the full GSPNR model. In similar manner as previously defined with GSPNR models, suppose $\mathcal{DA} = (D, A, E)$ is a *DecAct-G* representing a multi-robot system with $n$ different robot types, $R = \{r_1, \cdots, r_n\}$. The decision node set $D$ must be partitioned into $n$ subsets: $D = \bigcup_n D_i$, each being mutually exclusive $\forall i, j : D_i \cap D_j = \emptyset$. A decision node $d$ belonging to subset $D_i$, $d \in D_i$ is associated with robot type $r_i \in R$. Furthermore, cooperation actions involving two or more robots can be represented using *DecAct-G*s. An action done in cooperation by multiple robots is represented by an action node with multiple decision edges. Each decision edge connected to this cooperation action node represents a single robot that is involved in cooperatively executing the action. As with homogeneous systems, each decision edge that connects a decision node $d_1 \in D$ to action node $a \in A$, must have a matching outcome edge connecting the action node to a decision node $d_2 \in D$. When working with a heterogeneous system, these two decision nodes must belong to the same type subset: $d_1, d_2 \in D_i$. This corresponds to respecting the constraint of keeping the amount of each robot-type constant.

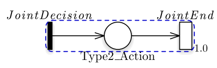
Actions done with cooperation require to define as many action models as there are robots executing the action. We considered synchronized cooperative actions, where all robots start and end the execution of the action at the same time. To model a synchronized cooperative action that is done in cooperation by $n$ robots, $n$ action places need to be used. Each action place corresponds to one of the robots executing its component of the cooperative action. Synchronized cooperative actions begin and finish at the same time. Thus, all action places are connected by an input arc to the same immediate transition. This immediate transition represents the decision to start the synchronized cooperative action. When the immediate transition fires, tokens are created in all action places and action execution begins. In similar manner, all action places are connected by an output arc to the same exponential transition. This constrains all robots to finish executing the action at the same time. When this transition fires, all tokens in the action places are consumed.

A synchronized action between $n$ robots requires $n$ separate action models. Every action model must contain the same *decision* and *final* transitions. Consequently, when these action models are merged, all action places are connected to same immediate or exponential transition, by an output or input arc, respectively. An example for a synchronized action done by two robots can be seen in Figure 3. Each robot type has its own action model for the same action $a$, comprising of its decision transition, action place, and final transition. The example considers two arbitrary robot types *'Type1'* and *'Type2'*. When transition "JointDecision" is fired, two tokens are created in each robot-type's action place. Finally, the two robots finish executing the synchronized action at the same time, and so both action places are connected to the same exponential transition "JointEnd".

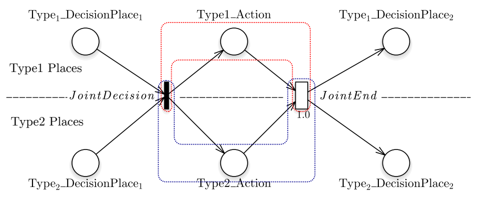
To build a heterogeneous GSPNR model from a *DecAct-G* decision nodes are interpreted as decision places and added to the GSPNR model. Unlike with homogeneous systems, in heterogeneous systems a single action node $a$ can have multiple action models (if the action is done in cooperation by multiple robots). All action models for each action node, $ActionModel(a, .)$ are instantiated and merged into the GSPNR model. Afterwards, the algorithm connects the action models to the appropriate decision



(a) $ActionModel(a)$ for *Type1*



(b) $ActionModel(a)$ for *Type2*



(c) Complete model obtained by merging the two models for each robot type, connected to decision places before and after executing the joint action;

Figure 3: Separate and merged models for a cooperative synchronized action between two different robot types

places, according to the edges in the *DecAct-G*.

### 3.4. Modelling Attributes

#### 3.4.1 System Attributes

In order to allow a model of system resources or attributes that can condition the decision-making of the robots, a new type of place is used - *Resource places*. In our approach, until now, all tokens have represented robots, and the place where the token is specifies the state of the robot. Tokens in *resource places* do not follow this interpretation and they represent a particular component of the state of the environment. A direct consequence is that the sub-GSPNR composed of all resource places does not need to be conservative. The number of tokens in all resource places can increase or decrease. Nonetheless, this sub-GSPNR made up of *resource* places must still be bounded. We chose to distinguish *resource* places from *decision* and *action* places by adding the label **"r."** to any *resource* place.

When building a GSPNR model from a *DecAct-G*, resource places can be directly included in action models, and can connect to its immediate or exponential transition. By including the same general place in multiple action models, specifications involving different actions can be made.

#### 3.4.2 Robot-specific Attributes

Modelling a multi-robot problem with a GSPNR where robots are represented anonymously as tokens presents a difficult challenge when including a robot-specific attribute. The GSPNR formalism offers no manner of distinguishing between tokens, and this is essential when capturing the evolution of a robot-specific attribute. The model must distinguish between robots where the attribute is at different states. Our approach to solving this challenge is including information on the current attribute state of a robot within the GSPNR places. Up to now, the place in which a token is specifies the state of the robot, but to model an attribute, each place must also specify the specific state of the attribute.

The modelled attribute must have a discrete set of possible states. For example, if the robot's battery is modelled, it must be approximated into a finite set of states such as {B0, B1, B2}. The battery state "B0" would be a low battery level, "B1" a medium battery level, and "B2" an almost full battery. A coarser approximation would discretize the battery into only two states {discharged, charged} that indicates if the robot still has battery or if it is discharged.

Suppose a robot-specific attribute $L$ has $N$ discrete states: $L = \{l_1, \cdots, l_N\}$. Broadly speaking, our approach to capture the attribute's evolution is to replicate $N$ times all *decision* and *action* places that pertain to the robot type with an attribute. In this new GSPNR model, each of these places represents not only the state of the robot (in terms of decisions available and action execution, for *decision* and *action* places respectively), but also contains information about the particular state of the robot attribute that the robot has. For example, if a UGV robot had a battery discretized into three states {B0, B1, B2}, all of its decision places $d_i$ would be replicated three times $\{d_i^0, d_i^1, d_i^2\}$. A token in place $d_i^0$ would represent a robot with low battery, a token in place $d_i^1$ would represent a robot with medium battery, and a token in place $d_i^2$ would represent a robot with high battery level. The same reasoning applies to *action* places. By replicating the *action* places, each token represents a robot with a specific attribute level executing a particular action.

But it is not enough to replicate only the *action* places. The transitions that model the beginning and end of each action also need to replicated. Thus, every $ActionModel$ involving a robot with an attribute must to be replicated for each level the attribute can take. These need to be unique: the $ActionModel$ for executing the $Travel$ action at battery level "B2" cannot be the same as the one when battery level is "B1". The simplest way to achieve this is by using a template $ActionModel$. This template can be then copied and customized for each specific attribute level. This way, we avoid having to explicitly list each $ActionModels(a, r, l)$ for each possible attribute level.

After a robot with an attribute finishes executing an action, its attribute may evolve to a different state. This is modelled with an *AttributeModel*. An *AttributeModel* is a GSPNR with a single place, de-

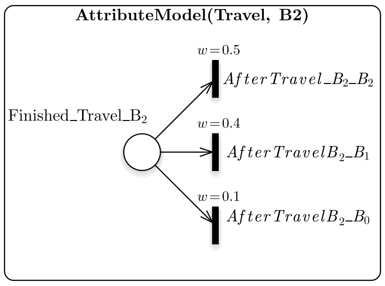


Figure 4: An attribute model for "Travel" action when the robot's battery has state "B2"

nominated *attribute* place. This place is connected to multiple immediate transitions, each modelling the event of the robot's attribute progressing to a different state. All of the immediate transitions in a *AttributeModel* can be associated with transitioning to a particular attribute state. For an attribute with $N$ possible states, $L = \{l_1, \cdots, l_N\}$, each *AttributeModel* will have at most $N$ immediate transitions $\{t_F^1, \cdots, t_F^N\}$. The transition $t_F^i$ is associated with the robot's attribute transitioning to state $l_i$.

Consider, for example, the *AttributeModel* depicted in Figure 4. This is the GSPNR that models the battery discharge for a robot that has just finished travelling having battery at state "B2". A token reaches the *attribute* place "Finished_Travel_B2", and the immediate transition that fires dictates which battery state the robot transitions to. If the robot's battery completely discharges and its battery transitions from state "B2" to "B0", the transition "AfterTravelB2_B0" fires, and the token representing the robot is consumed from the *attribute* place, and created in the decision place that represents the robot having battery "B0". The battery's state also has a chance of transitioning to level "B1", or staying at the same level "B2".

Considering an arbitrary action $a$, and an attribute with $N$ possible states, all of the *AttributeModels* can be specified as a series of conditional probabilities: $p(l_j|a, l_i)$. This expresses the probability of the robot's attribute transitioning to level $l_j$ after executing action $a$ and having started out at level $l_i$. An attribute model must be appended after any action model involving the robot with a modelled attribute. The transitions in the attribute model are then connected to the appropriate decision places, according to the outcome edges of the *DecAct-G*.

## 4. Multi-Robot GSPNR Toolbox

### 4.1. Overview and Architecture

We developed the software package as an open-source add-on toolbox for the MATLAB platform. The MATLAB platform was chosen for several reasons: 1) it has a large, established user community; 2) the company behind MATLAB provides long-term support and backwards compatibility as the platform is updated; 3) the simplicity of MATLAB's scripting language allows new users to take advantage of our toolbox without having having to learn complicated syntax.

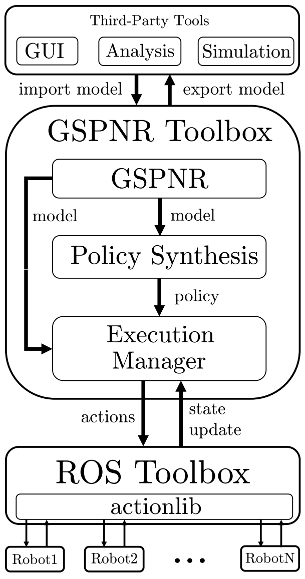


Figure 5: Overview of the toolbox's architecture

Conceptually, the toolbox can be divided into three separate modules, that can be used together or independently. All of these modules and their interactions can be seen in Figure 5. The main component is the *GSPNR* module, and it provides an implementation of GSPNR models and various ways to create and edit them. This module also allows the user to import and export these models from/to two external software packages: PIPE and GreatSPN. The *Policy Synthesis* module provides functions to compute optimal policies over GSPNR models using the value iteration algorithm on a GSPNR's equivalent MDP. The remaining module *Execution Manager* allows users to execute GSPNR policies on real robots. It does so by utilizing Matlab's ROS Toolbox, that allows communication with ROS action servers.

The package is available in a public repository [3]. It includes interactive tutorials (using the MATLAB's LiveEditor feature) that explain exactly how to use each of the three modules.

### 4.2. Computational Performance and Scalability

To test the computational performance of the toolbox and the scalability of each module, we devised a virtual multi-robot scenario where multiple robots operate in a indoors environment inside a house. The robots can travel between each subdivision of the house, and each of these rooms must be vacuumed and mopped by the robots. We ran two different experiments. The first experiment evaluated how the toolbox's performance scaled with modelling multi-robot problems where a fixed amount of robots could execute an increasing amount of actions. We did this by evaluating the performance of each module for a domestic scenario with an increasing number of rooms within the house. Adding rooms equates to allowing the robots to vacuum and mop more locations, and also adding possible navigation actions. The second experiment evaluated how the toolbox's performance scaled with modelling multi-robot problems with an increasing number of robots, for a fixed environment with the same number of locations.

For the *GSPNR* module, we measured how much time was taken to build the GSPNR model of the multi-robot problem, and the CPU and memory used. To characterize the *Policy Synthesis* module, we measured how much time time it took to build the equivalent MDP model, the number of states in this MDP model, and the CPU and memory used. Finally, the *Execution Manager* module was tested by executing a GSPNR plan for a duration of 5 minutes. We simulated the robots by creating mock-up *ActionServers* for the vacuuming and mopping action. Each time one of these *ActionServers* received a goal, it would block for a certain amount of time and then return the result message as if the robots succeeded carrying out the action.

All measures of CPU and memory usage were taken using an external Python script. This script measured the CPU and memory of the entire MATLAB process, which can have considerable overhead. CPU measurements were taken in relation to usage of a single core. Thus, 100% usage corresponds to fully utilizing one CPU core, 200% corresponds to fully utilizing two CPU cores, etc. On the other hand, all time measurements were done internally in MATLAB. The computer where all the tests were run has the following specifications: an i74690 CPU, 32GB RAM and a NVIDIA Geforce RTX 2080 Super graphics card.

We found no significant bottlenecks when creating ever larger GSPNR models, or models of larger robot teams. CPU and memory usage grew slowly and linearly as the number of locations in our model increased, and in particular CPU usage plateaued at 110% usage. The largest GSPNR model of a house with 42 different locations had 208 places and 374 transitions. When increasing the number of robots modelled, the CPU and memory used to create the GSPNR remained constant, and so did the time taken to create the GSPNR.

We did find a bottleneck when trying to synthesize policies over these models. The time taken to

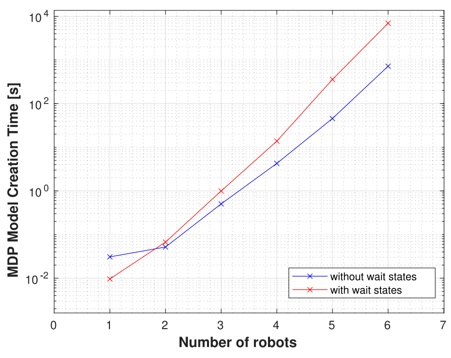


Figure 6: Time taken to create equivalent MDP with and without states as GSPNR model includes more robots;

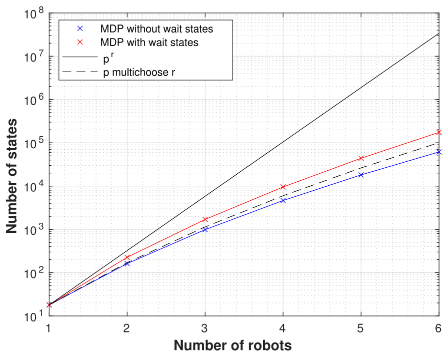


Figure 7: State space size of equivalent MDP with and without wait states as GSPNR model includes more robots;

convert a GSPNR model with an increasing amount of robots grows according to the graph in Figure 6. This is due to the size of the reachable marking set, which can be seen in Figure 7. As more robots are modelled, the corresponding state space also increases in an almost exponential manner. However, there is still a substantial advantage in modelling a multi-robot system with a GSPNR where robots are represented anonymously. Suppose we have a system with $r$ robots, and each robot can occupy $P$ different states. If the team's state is the concatenation of each robot's state $s_{team} = (p_1, p_2, ..., p_r)$, with $p_i \in P$, the maximum state space size is $P^r$. The GSPNR model of the exact same multi-robot scenario consists of $P$ places, each representing a robot's local state. The maximum number of reachable markings in such a GSPNR is "$P$ multichoose $r$". This is confirmed by verifying the number of states of the equivalent MDP without wait states in Figure 7 (in blue). Even when including wait states, the GSPNR model still has a significantly smaller state space.

In regards to execution, the results were promis-

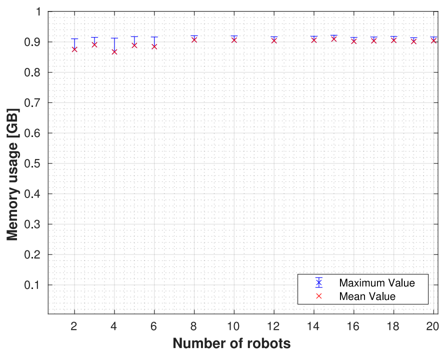


Figure 8: Running memory usage while executing GSPNR plan for team with up to 20 robots;

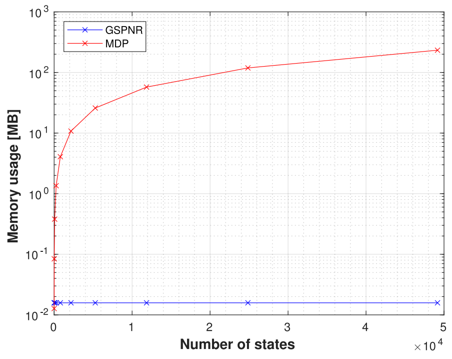


Figure 9: Size in megabytes of the `GSPNR` object (in blue) and the equivalent `MDP` object (in red) for an increasing amount of reachable states/markings;

ing. Even as more robots were coordinated, the computational requirements remained constant, as can be seen in Figure 8 for the running memory. Figure 9 further reinforces this point. For scenarios up to 10 robots, we measured the memory occupied by the MATLAB object that holds the GSPNR model and its equivalent MDP model without wait states. As the number of reachable states/markings increased, the GSPNR size stays constant, while the MDP model grows linearly. This is due to the fact that an MDP model has to contain all possible states, and all possible transitions for each state. A GSPNR model only holds the current state of the system, how the system evolves between states is encoded in its place/transitions/arcs structure.

## 5. Solarfarm Inspection Task Planning

The inspection scenario that was solved involved a set of 4 solar panels that need to be continuously inspected by two small, mobile robots with a limited battery life. In this application, there is no fixed location where the robots may recharge their battery, and so the team is augmented with a larger mobile robot that can recharge the inspecting agents. We assumed that this larger UGV has infinite battery-life and it can indefinitely recharge the smaller UGVs as many times as needed.

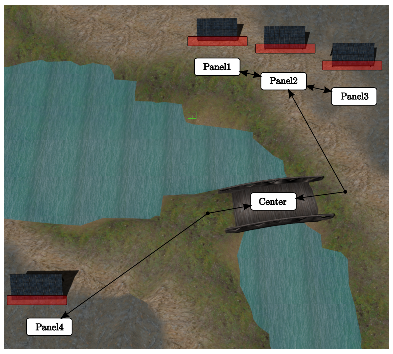


Figure 10: Gazebo world used in solarfarm simulation, with the topological map superimposed and the areas transversed by robots carrying out inspections outlined in red;

The Gazebo world simulating this scenario is depicted in Figure 10. It is based on the inspection world provided by Clearpath, but some modifications were done: the bridge was widened so that the warthog could pass through, terrain was smoothed to minimize slipping and some elements (such as the underground mine and the pipes) were removed.

Two *Jackal UGVs* were used to inspect the solar panels, and the already existing *multi-jackal* ROS package was used. The native localization (that fused GPS, IMU and odometry data) was turned off, and the ground truth data from the simulator was passed on to each jackal's move base so that they could localize themselves reliably. Each robot had their own simulated battery. This battery was deterministic, we did not include any type of noise when measuring the current battery level. These robots inspect each solar panel by navigating to a set of three positions nearby to the solar panel. The robots pause in each position to orientate themselves towards the solar panel, and then travel to the next position. This is repeated until 90 seconds have passed, and then the inspection is finished. To recharge cooperatively with the warthog, the jackals line up ahead of the warthog robot so that the jackal's charging plug lines up with the warthog's charging port. The jackal then backups into the warthog, and the simulated battery starts charging.

The existing *warthog* packages provided by Clearpath provided the necessary ROS interfaces to simulate a single larger UGV. These were adapted to use within a multi-robot problem, and a move base component was added to enable autonomous navigation. Localization was done the same way as the jackals, by passing on ground truth data from Gazebo to the move base node. The warthog's chassis itself was modified to include the charging port.

## 5.1. Results

The action duration parameters of the GSPNR model were estimated by running the navigation action server a single time for each navigation edge. We also estimated the mean charging time by running the charge action server a single time and recording how much time elapsed. For the battery discharge model, we chose simple parameters: every time a jackal finishes executing an action, it has 0.8 chance of staying at the same battery level, and 0.2 change of depleting its battery. We tested three different policies:

- **Random policy** - a policy where the system chooses to randomly fire a single transition from all available ones.
- **Handcrafted policy** - this policy was carefully designed by us, and it represents a greedy policy that tries to maximize the immediate number of inspections done. Both small UGVs try to inspect the closest available panel, and the larger UGV remains stationary until a small UGV is discharged. When this happens, the larger UGV navigates to closest discharged small robot and starts charging it.
- **Optimal policy** - this policy maximizes the *discounted expected reward.* We used a discount factor of $\gamma = 0.99$, and a convergence criterion of $\epsilon = 0.01$ to run the value iteration algorithm on the equivalent MDP without wait states. This MDP had 84,545 states/reachable markings.

In Gazebo, we did 8 one-hour runs of the three policies. The results can be seen in Figure 11, where the average accumulated reward is plotted as a function of execution time. The solid lines represent the average value, while the shaded areas represent the mean value plus and minus the measured standard deviation. The optimal policy was able to slightly outperform the handcrafted policy we designed. The handcrafted policy itself is good at coordinating the robots according to our goal specification, as it is a very large improvement over the random policy where no coordination is done. We also ran model simulation without robots to evaluate the accuracy of the our representation of uncertainty. Each GSPNR plan, for each of the 3 policies, was simulated in the model for 10 one-hour runs.

For each simulation, in both Gazebo and in the model simulation without robots, we measured three main metrics to classify the team's performance: the downtime percentage, the average time taken to inspect all 4 solar panels, and the average accumulated reward. The downtime percentage was measured as the proportion of running time where at least one robot was discharged and waiting for the warthog to be able to recharge. The average time between round of inspections was measured by counting the number of times the transition of the global counter "InspectedAll" fired, and dividing it by the running time.

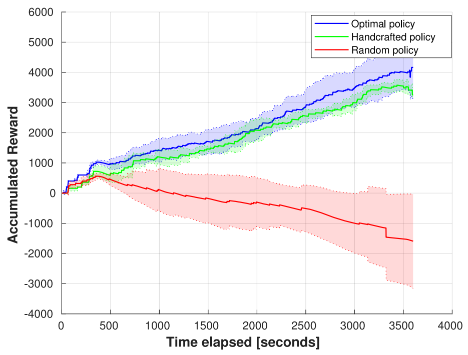


Figure 11: Average accumulated reward in Gazebo simulation in function of time, for three different policies: the **optimal policy** (in blue), the **handcrafted policy** (in green), and the **random policy** (in red).

These results can be seen in Table 1. The GSPNR plan simulated directly in the model without robots was a good predictor for the metrics measured in the real system simulated in Gazebo. More importantly, even though the optimal policy induces a larger downtime percentage when compared with the handcrafted policy, it was able to coordinate the inspections of the solar panels by the small robots better. The overall time taken to inspect all solar panels was shorter and on average it was also able to gain a greater total accumulated reward over 1 hour of execution time.

## 6. Conclusions

This main objective was to solve a multi-robot inspection task planning problem using a heterogeneous robot team. To achieve this, we developed a generic framework to model multi-robot heterogeneous task planning problems as a GSPNR. We extended previous work where robots were represented as tokens to be able to model heterogeneous teams, and we provided a way to include a robot's battery in the GSPNR model.

The toolbox developed is an open-source MATLAB add-on where users can easily create GSPNR models from scratch. We included a component that is able to synthesize optimal policies over these models. Lastly, we came up and implemented an algorithm to execute GSPNR plans that use these policies on real robotic systems, by integrating with the popular, open-source software library ROS. We characterized the toolbox's computational performance and scalability. The GSPNR modelling

| | GSPNR Plan with optimal policy | | GSPNR Plan with random policy | | GSPNR Plan with handcrafted policy | |
|---|---|---|---|---|---|---|
| | Model | Gazebo | Model | Gazebo | Model | Gazebo |
| # of runs | 10 | 8 | 10 | 8 | 10 | 8 |
| *downtime* [%] | 42.84 | 33.94 | 94.53 | 80.87 | 40.25 | 31.45 |
| $\Delta_{InspectAll}$ [s] | 528 | 614 | 3155 | 3136 | 636 | 642 |
| $\bar{r}$ [1/s] | 1.3484 | 1.1074 | -0.5756 | -0.3622 | 1.0888 | 1.0094 |

Table 1: Results obtained for executing GSPNR plans in model simulation without robots, and with robots simulated in Gazebo simulator. *downtime* is the proportion of time, where at least one robot was waiting to be charged, $\Delta_{InspectAll}$ is the average time between inspecting all 4 panels, and $\bar{r}$ is the average accumulated reward.

framework is scalable, and our extensions to heterogeneous systems and the ability to model robot-specific attributes allows the representation of very complex problems. Our implementation is also efficient and it allows users to save GSPNR models that have millions or billions of unique reachable states. Furthermore, our algorithm to execute GSPNR plans scales well as the number of robots coordinated grows, and it does not consume more computational resources as the executed models become increasingly more complex.

We finally solved the inspection problem. We modeled a heterogeneous robot team composed of three robots: 2 small UGVs and a single larger UGV. We approximated the small UGVs' battery into two discrete levels, and allowed the large UGV to cooperatively recharge the small UGVs. The toolbox we developed was able to create the GSPNR model for this multi-robot problem, and the optimal policy we obtained was able to outperform a carefully handcrafted strategy which was itself a large improvement over an uncoordinated system. This demonstrated the ease-of-use of our toolbox in creating these complex models. Additionally, it provided a robust execution algorithm whereby in the 24 hours of executing GSPNR task plans, it was always reliable and it correctly coordinated the robot team.

## Limitations and Future Work

The main limitation of the approaches and the toolbox we developed is associated with synthesizing optimal policies over the GSPNR models. The reachable marking set grows according to the expression "$P$ multichoose $r$", where $P$ is the number of states each robot can have, and $r$ is the number of robots modelled. Even though this is an improvement over individually modelling each robot, where the maximum number of states grows exponentially with the number of robots, it still severely limits the size of the multi-robot scenarios where optimal policies can be synthesized in feasible time. Any future work must tackle this limitation in order to be able to apply these approaches to large robot teams.

An alternative that might mitigate the state space problem are *approximated approaches* that use heuristics to go through the state space in a more efficient manner. However, policies obtained with these methods are not necessarily optimal or close to optimal. Additionally most of the heuristics used are not domain independent, having to come up with a good heuristic for each new problem approached. Still, the ability of our modelling framework to model a complex multi-robot problem, and the capacity of our toolbox to save and simulate these very large models can be used to sample the state space in model-based Reinforcement Learning methods. This becomes particularly useful when simulation of the system us not possible, and real-world data is scarce.

**Acknowledgements**

The author would like to acknowledge the funding received from Mathworks that specifically supported the work done in Section 4, through a joint Mathworks-RCF project, obtained via application.

## References

[1] Masoumeh Mansouri, Bruno Lacerda, Nick Hawes, and Federico Pecora. Multi-robot planning under uncertain travel times and safety constraints. In *The 28th International Joint Conference on Artificial Intelligence (IJCAI19), August 10-16, Macao, China*, pages 478–484, 2019.

[2] Carlos Azevedo, Bruno Lacerda, Nick Hawes, and Pedro Lima. Long-run multi-robot planning under uncertain action durations for persistent tasks. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4323–4328. IEEE.

[3] `https://github.com/cazevedo/multi-robot-gspnr-toolbox`. Last visited: 25/10/2021.