Under consideration for publication in Formal Aspects of Computing

Collaborative Models for Autonomous Systems Controller Synthesis

Douglas Fraser[1], Ruben Giaquinta[1], Ruth Hoffmann[2], Murray Ireland[3], Alice Miller[1] and Gethin Norman[1]

[1]School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, UK
[2]School of Computer Science, University of St Andrews, St Andrews, KY16 9SX, UK
[3]Craft Prospect Ltd, The Tontine, 20 Trongate, Glasgow G1 5ES

Abstract. We show how detailed simulation models and abstract Markov models can be developed collaboratively to generate and implement effective controllers for autonomous agent search and retrieve missions. We introduce a concrete simulation model of an Unmanned Aerial Vehicle (UAV). We then show how the probabilistic model checker PRISM is used for optimal strategy synthesis for a sequence of scenarios relevant to UAVs and potentially other autonomous agent systems. For each scenario we demonstrate how it can be modelled using PRISM, give model checking statistics and present the synthesised optimal strategies. We then show how our strategies can be returned to the controller for the simulation model and provide experimental results to demonstrate the effectiveness of one such strategy. Finally we explain how our models can be adapted, using symmetry, for use on larger search areas, and demonstrate the feasibility of this approach.

Keywords: Autonomous systems, formal verification, probabilistic model checking, strategy synthesis

1. Introduction

Autonomous vehicles such as unmanned aerial vehicles (UAVs), autonomous underwater vehicles and autonomous ground vehicles have widespread application in both military and commercial contexts. Investment in autonomous systems is growing rapidly and the UK government is investing £100 million into getting driverless cars on the road [Tre17], while the worldwide UAV market is expected to reach $21.5 billion by 2021 [EP18]. The U.S. Office of Naval Research has demonstrated how a swarm of unmanned boats can help to patrol harbours [Hsu16], the Defence Advanced Research Projects Agency (DARPA) has launched a trial of the world's largest autonomous ship [Rot16] and NASA has deployed Mars Rovers which, on receipt of instructions to travel to a specific location, must decide on a safe route [Ack13].

Understandably, there are concerns about safety and reliability of autonomous vehicles. Recently researchers exposed design flaws in drones by deliberately hacking their software and causing them to crash [Wil16], and US regulators discovered that a driver was killed while using the autopilot feature of a Tesla car

Correspondence and offprint requests to: Alice Miller, E-mail: alice.miller@glasgow.ac.uk

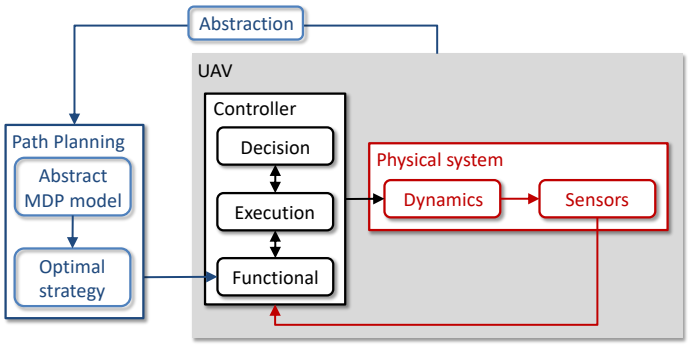


Fig. 1. Integrating optimal search strategies.

due to the failure of the sensor system to detect another vehicle [YT16]. Caltech's entry (Alice) in the 2007 DARPA Urban Challenge (DUP) for autonomous navigation within an urban environment was disqualified due to a problem that had not been discovered previously, despite thousands of hours of simulation and over three thousand miles of field testing [BDH+07]. Incidents like these are due to the difficulty of verifying highly complex systems with a tight coupling between computations and the physics of the environment. Individual components of such systems can be tested or formally verified, but systems as a whole are too intricate for either approach to be feasibly applied [KWT11].

Guaranteeing reliability of autonomous controllers using testing alone is infeasible, e.g. [KP16] concludes that autonomous vehicles would need to be driven hundreds of billions of miles to demonstrate their reliability and calls for the development of innovative methods for the demonstration of safety and reliability that could reduce the burden of testing. Formal verification offers hope in this direction having been used both for controller synthesis and for verifying the reliability and safety of autonomous controller logic.

Autonomous systems are implemented using software *controllers* that pre-determine the behaviour of the agent under a given set of internal parameter values and environmental conditions. In this paper we investigate the use of probabilistic model checking and the probabilistic model checker PRISM for automatic controller generation. Our ultimate goal is to develop software, based on the techniques described here, that can be embedded into controller software to generate *adaptable* controllers that are *verified* to be *optimal*, *safe* and *reliable* by design.

Modern controllers have two hierarchical layers [VNE+01]: the *functional* layer containing action and perception capabilities and the *decision* level which plans, oversees and controls the system's execution. Failures may come from either of these layers: verifying what happens at one level is not sufficient if the other layer is unreliable. Even though correct behaviour of individual components at either layer may be determined through exhaustive testing and the application of formal verification, the sheer complexity arising from the system as a whole makes complete verification impossible [KWT11]. We can only hope to build software from provably correct parts and rely on integration tools to ensure a satisfactory level of correctness of the overall system.

Our approach for generating guidance controller software for autonomous systems is illustrated in Fig. 1. An abstract Markov model is developed to represent the decisions controlling the search pattern of an autonomous system. The model checker, in this case PRISM, produces an optimal strategy with respect to a defined *cost*, such as expected mission time or probability of failure. This strategy is then incorporated into the functional level of the controller of an autonomous system, a UAV. The UAV uses a search pattern at the functional level to complete its mission, while the decision level of the control hierarchy handles when to interrupt the search mode, based on perception of the environment and mission targets, accomplished via onboard sensors. The goal is that this will lead to a lower cost of operation – such as a shorter mission time, or increase mission success when compared to a static search pattern.

To summarise, in this paper we:

1. introduce a concrete simulation model of a UAV;
2. describe abstract Markov models for a suite of scenarios inspired by situations relevant to UAVs and other autonomous agents modelled in the PRISM language;
3. present synthesised (optimal) strategies for the different scenarios and examine their performance;
4. demonstrate how the synthesised strategies can be returned as controllers for the simulation model;

5. provide experimental results demonstrating the effectiveness of the controllers;
6. present an approach to extend the work to larger search areas, and provide experimental results for this approach;
7. discuss the limitations of our approach and the next steps to overcome them.

This work is an extension of the conference paper [GHI+18] in which we describe a suite of abstract Markov models for strategy synthesis. This extension includes: full details of a concrete simulation model (which was originally used to derive parameters for the first of our Markov models, and is referred to in [GHI+18]); a description of how optimal strategies generated from the abstract MDP models can be returned to the concrete simulation model, together with experimental results; and an approach (together with experimental results) that uses symmetry to extend the work to generate controllers for searching larger areas.

Related Work. There have been many attempts to use formal verification for the verification and specification of robotic systems, see e.g. [LFD+18] for a survey. Our work concerns: the use of discrete-time abstract models for controller synthesis; decision making for UAVs; and the link between simulation and model checking. We therefore concentrate on these three aspects.

There has been significant work on using discrete-time abstract models, temporal logic specifications and formal methods for generating controllers of autonomous systems. These works differ in the temporal logic specifications and models used. These include approaches using the branching time logic PCTL [LAB15], linear time temporal logic LTL [WTM12, DSBR14], metric temporal logic [FT15], rewards [SM17], multi-objective queries [LK16, LPH17] and stochastic games [DFK+15, SKC+17, FWHT15]. The aim of these papers is the development of the required theory, rather than its application to a real system, which is the focus of our work. Also using discrete-time models are [Sha14, SCL+15], where partially observable MDPs (POMDPs) and LTL are used to generate motion plans for autonomous agents. Unlike our approach, the focus here is on developing efficient algorithms for solving POMDPs.

On the other hand, research using continuous-time models for generating controllers includes [BBFL18], where energy timed automata and Uppaal-Tiga [BCD+07], an extension of the Uppaal model checker for timed automata [LPY97], are used for the synthesis of controllers for resource-aware systems. There are also a number of works using continuous-time models to formally verify the control software for a UAV. For example, in [LKM+08] a continuous-time model is used to describe the dynamics of a UAV and is then used to verify the control system. Model checking of a continuous-time system is challenging, as the state-space of a continuous system is infinite in size. The use of hybrid automata is one solution to this problem [Hen96, CK00, KEPS99], however the formal verification techniques which are applicable to such a model are computationally expensive. To overcome this limitation, [FDW13, DFL+16] develop a compositional based approach, where the discrete decision-making components can be analysed separately from the continuous hybrid aspects. In this paper we also separate components, but use model checking for controller synthesis rather than verifying existing controllers.

Related work in which abstract models are derived from concrete simulation models for verification include the formal analysis of Simulink models [DH04]. For example, [MBR06, BBB+12] verify discrete time Simulink blocks describing avionics/aerospace components using the NuSMV [CCGR99] and DiVinE [BBH+13] model checkers respectively. In [Mil09], Simulink components are translated to the Lustre formal specification language [HCRP91], from which they can be verified using a variety of model checkers and theorem provers, while [ASK04] presents a translation from Simulink to hybrid automata. An automatic translation of real-time Simulink blocks to the input language of the Uppaal statistical model checker Uppaal-SMC [DDL+12] is used to verify two automotive systems in [FMM+16].

Alternative approaches in this area include [MMBC11, JYL+16], which translate Stateflow diagrams to timed and hybrid automata respectively to allow formal verification. The first of these includes a discrete abstraction-based algorithm for synthesizing supervisory controllers, and the second uses Uppaal to verify aspects of a train controller system. Recently, an automatic and sound translation of robotic models specified in the $G^{en}_{\circ}M3$ robotic framework to Uppaal-SMC [DDL+12] for the verification of an autonomous UAV was considered in [FIS19]. In their approach templates representing functional components are formalised as timed transition systems (TTSs). These are translated to timed automata augmented with global urgencies and data (DUTA) which can then be automatically mapped into Uppaal and Uppaal-SMC. Their goal is the verification of real-time functional (schedulability and bounded response) properties rather than the generation of optimal strategies.

In this paper, not only do we use model checking to synthesise optimal decision strategies, but we return

them to the guidance controller software of a UAV. Although we use our simulation model to inform our abstract model design for our first scenario described in Section 4, the main contribution of this work is the transfer of optimal strategies *from* abstract MDP models *to* guidance controller software.

Paper Outline. We provide background material on MDPs and probabilistic model checking in Section 2. In Section 3 we describe our concrete simulation model. We present a range of scenarios relevant to autonomous agents in Section 4 and demonstrate how PRISM can be used for verification and strategy synthesis. Section 5 addresses the issue of how our synthesised strategies can be returned to the MATLAB model, providing implementation detail and experimental results. We also explore how the approach can be widened to a larger search area using symmetry. Our conclusions are presented in Section 6.

2. Background

We now introduce Markov decision processes (MDPs) and probabilistic model checking of MDPs in PRISM. For any finite set X, let $Dist(X)$ denote the set of discrete probability distributions over X.

Markov Decision Processes. MDPs model discrete time systems that exhibit both nondeterministic and probabilistic behaviour.

Definition 2.1. A *Markov decision process* (MDP) is a tuple $\mathsf{M}=(S, \bar{s}, A, P, L)$ where:

- S is a finite set of states and $\bar{s} \in S$ is an initial state;
- A is a finite set of *actions*;
- $P : S \times A \to Dist(S)$ is a (partial) probabilistic transition function, mapping state-action pairs to probability distributions over S;
- $L : S \to 2^{\mathrm{AP}}$ is a labelling function assigning to each state a set of atomic propositions from a set AP.

In a state s of an MDP M, there is a nondeterministic choice between the *available* actions in s. These available actions, denoted $A(s)$, are the actions for which $P(s, a)$ is defined. If action a is selected, then the successor state is chosen probabilistically, where the probability of moving to state s' is $P(s, a)(s')$. An execution of an MDP is a path corresponding to a sequence of transitions of the form $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$, where $a_i \in A(s_i)$ and $P(s_i, a_i)(s_{i+1}) > 0$ for all $i \geqslant 0$. Reward structures model quantitative measures of an MDP which are accumulated when an action is chosen in a state.

Definition 2.2. A *reward structure* for an MDP $\mathsf{M}=(S, \bar{s}, A, P, L)$ is a function of the form $r : S \times A \to \mathbb{R}_{\geqslant 0}$.

To reason about the behaviour of an MDP, we need to introduce the concept of *strategies* (also called policies, adversaries and schedulers). A strategy resolves the nondeterminism in an MDP by selecting the action to perform at any stage of execution. In general, the choice of the action can depend on the history and be made randomly, however for the properties we consider, deterministic and memoryless strategies are sufficient.

Definition 2.3. A (deterministic and memoryless) *strategy* of an MDP M is a function $\sigma : S \to A$ such that $\sigma(s) \in A(s)$ for all $s \in S$.

Under a strategy σ of an MDP M, the nondeterminism of M is resolved, and hence its behaviour is fully probabilistic. Under a fixed strategy, the behaviour of an MDP corresponds a discrete time Markov chain (DTMC). We can use a standard construction on DTMCs [KSK76] to build a probability measure over the infinite paths of M.

Property specifications. Two standard classes of properties for MDPs are *probabilistic* and *expected reachability*. For a given atomic proposition (or conjunction of propositions), these correspond to the probability of eventually reaching a state labelled by the proposition (or propositions) and the expected reward accumulated before doing so. The value of these properties depends on the resolution of the nondeterminism, i.e. the strategy, and we therefore consider optimal (minimum and maximum) values over all strategies.

The probabilistic model checker PRISM. PRISM [KNP11] is a probabilistic model checker that allows for the analysis of a number of probabilistic models including MDPs. Models in PRISM are expressed using a high level modelling language based on Reactive Modules [AH99]. A model consists of a number of interacting modules. Each module consists of a number of finite-valued variables corresponding to the module's state and the transitions of a module are defined by a number of guarded commands of the form:

$$[\texttt{<action>}]\ \texttt{<guard>}\ \rightarrow\ \texttt{<prob>} : \texttt{<update>} + \cdots + \texttt{<prob>} : \texttt{<update>}$$

A command consists of an (optional) action label, guard and probabilistic choices between updates. A guard is a predicate over variables, while an update specifies, using primed variables, how the variables of the module are updated when the command is taken. Interaction between modules is through guards (as guards can refer to variables of all modules) and action labels which allow modules to synchronise. Support for rewards are through reward items of the form:

$$[\texttt{<action>}]\ \texttt{<guard>} : \texttt{<reward>};$$

representing the reward accumulated when taking an action in a state satisfying the guard.

PRISM supports the computation of optimal probabilistic and expected reachability values, for details on how these values are computed and the temporal logic that PRISM supports, see [FKNP11]. PRISM can also synthesise strategies achieving such optimal values. For the properties we consider, the synthesised strategies are deterministic and memoryless, and therefore can be represented as a list of (optimal) action choices for each state of the MDP under study. This list can then be imported back into PRISM to generate the underlying DTMC, and hence allow further analysis of the strategy. For details on strategy synthesis see [KP13]. In PRISM, the minimum and maximum probability of eventually reaching a state labelled by the atomic proposition `a` are expressed by the temporal logic formulae $\texttt{P}_{\text{min=?}}[\,\texttt{F a}\,]$ and $\texttt{P}_{\text{max=?}}[\,\texttt{F a}\,]$ respectively, and the minimum and maximum expected reward accumulated according to the reward structure `r` before reaching a state labelled by `a` are given by $\texttt{R}^{\texttt{r}}_{\text{min=?}}[\,\texttt{F a}\,]$ and $\texttt{R}^{\texttt{r}}_{\text{max=?}}[\,\texttt{F a}\,]$ respectively.

3. MATLAB Model of a UAV

We employ a simple scenario where a quadrotor UAV is in operation inside a small, constrained environment. This environment is based on the University of Glasgow's Micro Air Systems Technologies (MAST) Laboratory, a 4×7×3 m cuboidal flight space with an Optitrack[1] motion capture system for tracking UAVs. Three inanimate target objects, each of unique colour and shape, are located somewhere on the floor of the lab. The UAV is programmed to take off from a specified landing site and follow a predetermined series of waypoints, while searching for the objects. On finding a target, the quadrotor retrieves it, transports it to a fixed drop site and deposits it. It then continues following the waypoints until all of the targets have been located or it reaches the final waypoint, then finally returns to base. Stochastic behaviour is introduced through the inclusion of faults, and the initial locations of the targets, landing site and drop site.

3.1. Mathematical Model

The mathematical model describes a multi-agent system, comprising the UAV, targets and environment, and is implemented in MATLAB® using an object-oriented programming approach. The quadrotor may be considered as either an *active* or *cognitive* agent [KMP10]. The UAV has guidance software containing both a flight control system and decision-making algorithms. It is these decision-making algorithms that allow the UAV to perform its mission with no interference from a human operator. The targets are considered as *passive* agents [KMP10], in that they have their own dynamics, but no goals or reactive behaviours. All agents exist within an environment, which corresponds to the MAST Laboratory.

Fig. 2 shows a simplified diagram of the multi-agent system. The UAV's sensor suite senses both the internal states of the quadrotor and the geometry of the targets and environment. The sensor measurements are utilised by the guidance system of the UAV, which then provides control inputs to the UAV's rotors. The guidance system comprises several subsystems which allow the UAV to interpret the sensor feedback, decide on a course of action and drive the vehicle towards that action.

Quadrotor Dynamics. Quadrotor dynamic models are well-documented in the literature [BMS04, Voo09, Ire14, TM06] and will not be replicated here, except where some additions have been made. For the purposes of creating a realistic but otherwise simple simulation, some assumptions and approximations have been made in implementing the model. These are stated where relevant. In general, the quadrotor is described by a non-linear, 6 degree-of-freedom model. Control of the quadrotor is achieved via four rotor speed commands,

[1] Natural Point, Inc. www.optitrack.com

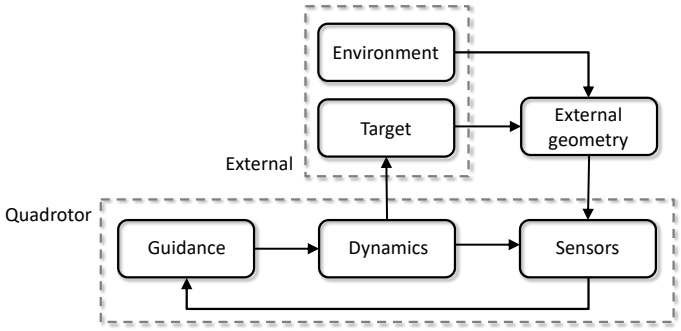


Fig. 2. Simplified representation of a multi-agent system.

while internal and external sensors provide accurate measurements of the quadrotor position, orientation and velocity. For the purposes of the simulation, state recreation for feedback control is assumed to be highly-accurate and any latency between the sensor measurements and their usage in the control system neglected. For the benefit of aiding later description of feedback control laws in this paper, the quadrotor model may be described by the general form:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \quad \text{and} \quad \mathbf{y}(t) = h(\mathbf{x}(t), \mathbf{u}(t)) \tag{1}$$

where t is time, $\mathbf{x}$ is the continuous state vector, $\mathbf{u}$ is the input vector and $\mathbf{y}$ is the output vector. f and h are non-linear functions of the state and input. The state vector comprises the position $\mathbf{r}_Q$ and orientation (or attitude) $\boldsymbol{\eta}_Q$ of the quadrotor and the rates of change of position and orientation.

Grasper Arm Dynamics. When a target is tethered to the quadrotor, its dynamics are coupled to those of the grasper arm. The dynamics of the arm are thus considered separately from the UAV's rigid-body response. The grasper has fixed position $\mathbf{r}^{\mathcal{Q}}_{G/Q} \in \mathbb{R}^3$ in the rotating frame $\mathcal{Q}$. It therefore has the inertial position and second derivative:

$$\mathbf{r}_G = \mathbf{r}_Q + \mathbf{R}^{\mathcal{W}}_{\mathcal{Q}} \mathbf{r}^{\mathcal{Q}}_{G/Q} \quad \text{and} \quad \ddot{\mathbf{r}}_G = \ddot{\mathbf{r}}_Q + \mathbf{R}^{\mathcal{W}}_{\mathcal{Q}} \left[\boldsymbol{\omega}_Q \times \left(\boldsymbol{\omega}_Q \times \mathbf{r}^{\mathcal{Q}}_{G/Q} \right) + \dot{\boldsymbol{\omega}}_Q \times \mathbf{r}^{\mathcal{Q}}_{G/Q} \right] \tag{2}$$

where translational and rotational accelerations, $\ddot{\mathbf{r}}_Q$ and $\dot{\boldsymbol{\omega}}_Q$ respectively, describe the motion of the quadrotor and are detailed in the aforementioned references [BMS04, Voo09, Ire14, TM06].

Gimbal Dynamics. The camera sensor is mounted on a gimbal which stabilises it during rolling and pitching. See [HIM+16] for further details.

Battery Model. A battery model is employed to allow us to introduce the complication of the UAV periodically returning to base to recharge. As the battery discharges, the voltage supplied to the UAV varies as described by the first-order model:

$$\dot{V} = \begin{cases} 0 & \text{if charging and } V \geqslant V_{\max} \\ 0 & \text{if not charging and } V \leqslant 0 \\ v_c & \text{if charging and } V < V_{\max} \\ v_d & \text{if not charging and} V > 0 \end{cases}$$

where $v_c > 0$ is the charging rate, which applies when the UAV is idle at the landing site, and $v_d < 0$ is the discharge rate, which applies when the UAV is attempting to carry out its mission. The voltage level has the upper limit $V_{\max}$, above which it will not charge. The discharge rate is approximated as a constant. In reality, the battery drain is affected by the power requirements of the rotors and other hardware components.

Onboard Camera Model. The *Camera Sensor* provides an image of any unobscured object within its field of view. In reality, the camera captures an image which is then used in an object detection algorithm. See [HIM+16] for further details. Visual target detection allows vision-based navigation by providing target coordinates to the real-time control system.

Target Model. While static for the majority of the mission, each target is given simple dynamic behaviours in order to facilitate its interaction with the UAV. Should the UAV drop the target object (due to a grasper

fault or return-to-charge override), the target will fall to the ground and remain in this location until the UAV encounters it again.

3.2. Stochastic Properties

Our simple deterministic quadrotor model has enough detail that it may realistically describe a quadrotor which is capable of sitting idle on the ground, flying and grasping objects. In reality, the quadrotor is subject to a number of stochastic events which can impact the mission. Here, we discuss such events and, if relevant, how we incorporate them into our model.

External disturbances are products of the local environment and are governed by a system far greater in complexity than the quadrotor itself. Thus, we may consider them to be random. These disturbances may manifest as forces or moments acting on the vehicle body and may impact the performance of the rotors by changing the local airflow around the rotor disk. A robust control system is typically employed to reduce the impact of such disturbances. In reality, even with robust control, these disturbances may have the effect of changing the path the UAV takes, or the time taken to perform a manoeuvre.

The quadrotor system may also be subject to *internal disturbances*. These may manifest as a change in a parameter which is expected to be constant. For example, the rotor thrust gain can be altered by a number of phenomena, including varying battery level and local airflow around the rotor, as indicated above. One or more rotors may also fail entirely, due to a fault in any of the components between autopilot and rotor, such as the motor or propeller. Regardless of where the fault occurs, the effect is a complete loss of thrust and torque from the rotor. We model such an *actuator fault* as a complete loss of thrust in a single rotor, and assume that the probability of such an event has a geometric distribution. While control strategies are available to deal with such a fault [MD14], the reduced capability of the vehicle typically results in a controlled emergency landing, if not a crash landing. Our control algorithm specifically includes instructions for emergency landings after actuator loss.

A *fault in the grasping mechanism* may result in the mechanism becoming stuck or releasing any tethered objects. We limit our model to include the latter case and again assume the probability is modelled using a geometric distribution. It is assumed that a fault causes the grasper mechanism to deactivate. This has no effect when no target is tethered. However, if a target is tethered, it is released and the UAV must retrieve it again to continue the mission. If a target is released, then it follows a ballistic trajectory before colliding with the ground. In this phase, the state transition of the target is purely deterministic.

We randomly define the *pose* (position and heading) of the UAV at the beginning of the mission as well as the initial position of each target and the drop site. This has the effect of altering the behaviour of the UAV as the predicates which trigger transitions from one mode to another may become true at different times and under different circumstances.

3.3. Autonomous Guidance System

The guidance system of the UAV consists of a number of autonomous *modes* which dictate the behaviour of the UAV at any given time. Each mode comprises a combination of automatic control and trajectory modules, which collaborate to drive the UAV towards a specific position and heading. The modes and the transitions between them are represented in the finite state machine (FSM) shown in Fig. 3. The UAV can respond to any internal or external event considered in the simulation. We now detail the behaviour of each mode with reference to the control and trajectory modules or commands employed in each case. For clarity, a list of model symbols are provided in Table 1.

- *Idle.* The UAV sits at rest with its rotors inactive. The UAV begins the mission in this mode and exits this mode only if the mission is not completed and the battery is fully charged, and progresses to *Take-off*.
- *Take-off.* The UAV takes off to the position $\mathbf{r}_d = [x_0, y_0, z_{hvr}]^T$ above the landing site, where z_{hvr} is the hover height. This ensures that the ground effect region is cleared before performing further manoeuvres. It then proceeds to *Initialise* upon satisfying the condition $\|\mathbf{r}_Q - \mathbf{r}_d\| < tol$ where tol is a tolerance value.
- *Initialise.* The UAV performs self-diagnosis to identify any system faults. If a fault is detected, then the UAV proceeds to *Land*, otherwise the UAV transitions to *Search*.
- *Search.* The UAV follows a series of waypoints as defined by the *Search pattern trajectory* module.

Symbol	Description
I	Inertia tensor
K	Controller gains
R_Q	Max distance between quadrotor CG and body exterior
R_T	Max distance between target CG and body exterior
$\mathbf{r}_d$	Desired position vector
$\mathbf{r}_G$	Desired position vector
$\mathbf{r}$, $\mathbf{r}_Q$	Quadrotor position vector
$\mathbf{r}_T$	Target position vector
$\mathbf{u}$	Quadrotor input vector
$\mathbf{v}$, $\mathbf{v}_Q$	Quadrotor velocity vector
x, y, z	Elements in position vector $\mathbf{r}$
$\boldsymbol{\eta}$	Quadrotor orientation vector
$\boldsymbol{\omega}$	Quadrotor angular velocity vector
ϕ, θ, ψ	Elements in orientation vector $\boldsymbol{\eta}$

Table 1. List of mathematical model symbols, with application (best described in [Ire14]).

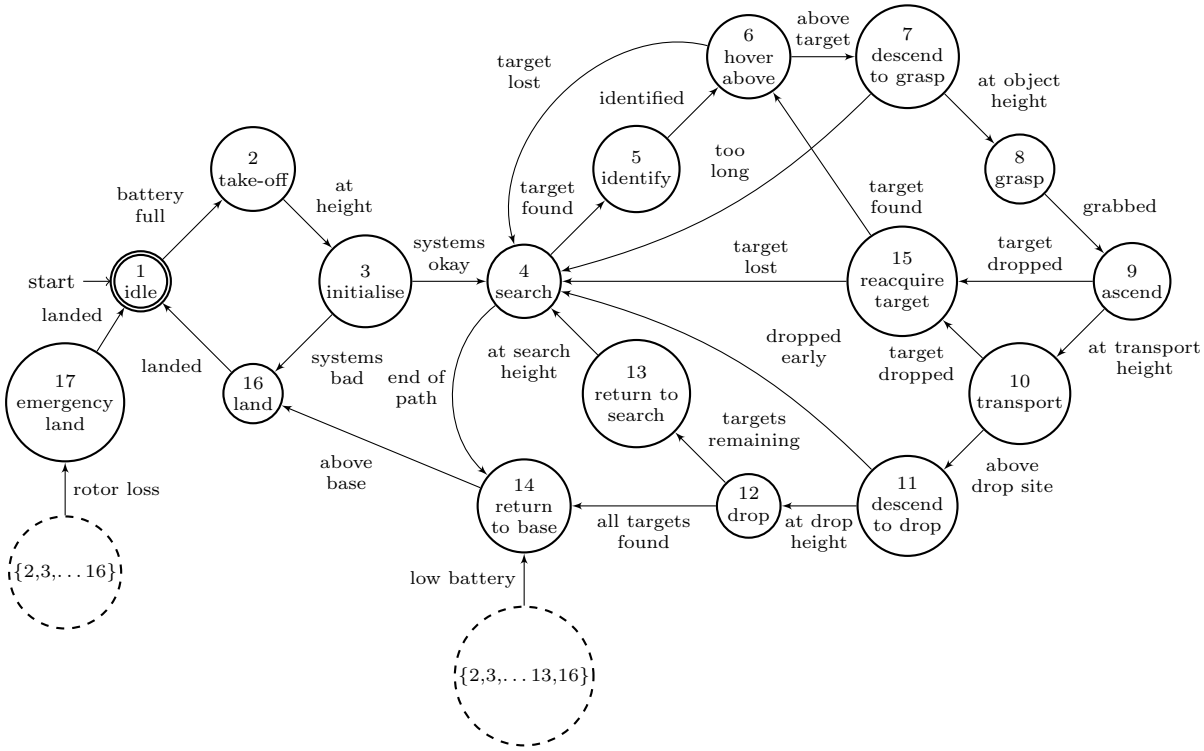


Fig. 3. Finite-state machine describing the AI logic.

The spacing of the waypoints and the search height z_{srch} are defined such that the camera's field of view overlaps as it passes between pairs of waypoints. The velocity command is limited to ensure the UAV does not pass over targets too quickly. The *Object tracking* module is active and provides a Boolean variable *found* which specifies whether a target has been detected. When *found* is set to `true` a transition to *Identify* is triggered. If the UAV reaches the final waypoint, then not all targets have been found and the UAV proceeds to *Return to base* and triggers a *Mission failed* flag. Reaching a time limit $T_{\max}$ defined for the mission triggers similar behaviour.

- *Identify.* The UAV moves to the *Identify* mode when a target is detected. The UAV typically has some lateral motion during this transition and the controller compensates for this by returning the UAV to the position recorded when it entered *Identify*, denoted $\mathbf{r}_{entry}$. Once it has returned within a small radius of this position, that is $\|\mathbf{r}_Q - \mathbf{r}_{entry}\| < tol$, the UAV progresses to *Hover above target.*

- *Hover above target.* The UAV is positioned above the target which makes it visible to the camera sensor. The *Object tracking* module detects blocks of colour within a certain RGB range and determines the coordinates of their centroids. The coordinates of the centroid closest to the centre of the image are supplied to the *Visual controller* resulting the UAV moving directly above the target. When the vehicle velocity in the horizontal plane is near-zero the UAV proceeds to *Descend to grasp*.
- *Descend to grasp.* The UAV uses the *Object tracking* and *Visual controller* modules to maintain its position above the target, while descending. When the grasping mechanism position is within a small radius of the target, i.e. $\|\mathbf{r}_G - (\mathbf{r}_T - R_T)\| < tol$, the UAV transitions to *Grasp*.
- *Grasp.* The grasping mechanism is activated and the dynamics of the grasped target are tethered to those of the UAV. The UAV then proceeds to *Ascend*.
- *Ascend.* The UAV ascends to the transport height $z_{trnsprt}$, while maintaining zero velocity in the horizontal plane. The UAV proceeds to *Transport* upon reaching $z_{trnsprt}$, i.e. when $|z_Q - z_{trnsprt}| < tol$. If a grasping mechanism fault occurs, the UAV moves instead to *Reacquire target*.
- *Transport.* The UAV transports the tethered target to the position $\mathbf{r}_d = [x_{ds}, y_{ds}, z_{trnsprt}]^T$ above the drop site. The UAV proceeds to *Descend to drop* when $\|\mathbf{r}_Q - \mathbf{r}_d\| < tol$.
- *Descend to drop.* The UAV descends to a height where the target is touching the floor, i.e. to $z_d = -(z_{G/Q} + 2R_T)$. If the target is dropped early due to a grasping mechanism fault, the target is recorded as having been successfully deposited and the UAV proceeds to *Return to search*. Otherwise, it progresses to *Drop* upon satisfying $\|\mathbf{r}_Q - \mathbf{r}_d\| < tol$, where $\mathbf{r}_d = [x_{ds}, y_{ds}, z_d]^T$.
- *Drop.* The grasping mechanism is deactivated and the target is untethered from the UAV. The UAV then proceeds to *Return to search* if the mission is not complete (i.e. there are targets still to be found) or to *Return to base* if all targets have been deposited at the drop site.
- *Return to search.* The UAV ascends vertically to search height above the drop site, that is, the position $\mathbf{r}_d = [x_{ds}, y_{ds}, z_{srch}]^T$. It then proceeds to *Search* upon satisfying $\|\mathbf{r}_Q - \mathbf{r}_d\| < tol$.
- *Return to base.* From any mode, a transition to *Return to base* is triggered by a low battery warning with any tethered targets released. In this mode, the UAV returns to the position $\mathbf{r}_d = [x_0, y_0, z_{hvr}]^T$ above the landing site and transitions to *Land* upon satisfying $\|\mathbf{r}_Q - \mathbf{r}_d\| < tol$.
- *Land.* The UAV descends from its position above the landing site to the landing site position $\mathbf{r}_d = [x_0, y_0, -R_Q]^T$. The UAV then returns to *Idle* upon satisfying the conditions $\|\mathbf{r}_Q - \mathbf{r}_d\| < tol_1$ and $\|\dot{\mathbf{r}}_Q\| < tol_2$. This ensures the motors are set to idle when the UAV is stationary at the landing site.
- *Reacquire target.* A transition to *Reacquire target* occurs immediately when the grasping mechanism fails. As the UAV may have a high velocity when the transition occurs, it must return to the position $\mathbf{r}_{entry}$ at which it entered the *Reacquire target* mode in order to maximise the probability of visually reacquiring the target. To exit this mode, the conditions $\|\mathbf{r}_Q - \mathbf{r}_{entry}\| < tol_1$ and $\|\dot{\mathbf{r}}_Q\| < tol_2$ must be satisfied. The guidance mode changes to *Hover above target* if the *Object tracking* module detects a target and to *Search* otherwise.
- *Emergency land.* A transition to *Emergency land* may occur in any mode except *Emergency land* itself and *Idle*. The *Emergency controller* is employed, accepting only the height command $z_d = -R_Q$. The UAV returns to *Idle* when $|z_Q + R_Q| < tol$ is satisfied. This mode is also reached when an actuator fault is detected in any mode except *Idle*.

The control and trajectory modules active at any time are determined by the UAV's mode and specified in Table 2. The modules work in combination to follow the commands issued by the active mode. For example, in the *Take-off* mode the desired behaviour is to hover at a static position above the landing site, and therefore a constant position command is supplied to the *State feedback controller*. During *Search*, the UAV follows one waypoint after another. The *Search pattern module* issues changing position commands and the *State feedback controller* tracks them. A *State reconstruction module* is active at all times which reconstructs the dynamic states of the quadrotor from the available sensor measurements. The vehicle control laws determine the rotor inputs $\mathbf{u}$ using a feedback-linearised controller, described in [IVA15, Voo09].

State Reconstruction Module. The control and navigation systems of the UAV require knowledge of the system state $\mathbf{x}$. This is not directly obtainable and must be measured through the sensors. For the purposes of this simulation, it is assumed that the position $\mathbf{r}_Q$ and orientation $\boldsymbol{\eta}_Q$ are perfectly reconstructed from the measurements taken by an external motion capture system, while the angular rates $\boldsymbol{\omega}_Q$ are reconstructed

Mode	Input Command/Module	Trajectory Command/Module
1 Idle	$\mathbf{u}_Q = [0,0,0,0]^T$	–
2 Take-off	State feedback module	$\mathbf{r}_d = [x_0, y_0, z_{\text{hvr}}]^T$
3 Initialise	State feedback module	$\mathbf{r}_d = [x_0, y_0, z_{\text{hvr}}]^T$
4 Search	State feedback module	Search pattern module
5 Identify	State feedback module	$\mathbf{r}_d = \mathbf{r}_{\text{entry}}$
6 Hover above target	State feedback and visual modules	Object tracking module; $z_d = z_{\text{srch}}$
7 Descend to grasp	State feedback and visual modules	Object tracking module; $z_d = -(z_{G/Q} + 2R_T)$
8 Grasp	State feedback module	$\dot{x}_d = 0; \dot{y}_d = 0; z_d = -(z_{G/Q} + 2R_T)$
9 Ascend	State feedback module	$\dot{x}_d = 0; \dot{y}_d = 0; z_d = z_{\text{trnsprt}}$
10 Transport	State feedback module	$\mathbf{r}_d = [x_{ds}, y_{ds}, z_{\text{trnsprt}}]^T$
11 Descend to drop	State feedback module	$\mathbf{r}_d = [x_{ds}, y_{ds}, -(z_{G/Q} + 2R_T)]^T$
12 Drop	State feedback module	$\mathbf{r}_d = [x_{ds}, y_{ds}, -(z_{G/Q} + 2R_T)]^T$
13 Return to search	State feedback module	$\mathbf{r}_d = [x_{ds}, y_{ds}, z_{\text{srch}}]^T$
14 Return to base	State feedback module	$\mathbf{r}_d = [x_0, y_0, z_{\text{hvr}}]^T$
15 Reacquire target	State feedback module	$\mathbf{r}_d = \mathbf{r}_{\text{entry}}$
16 Land	State feedback module	$\mathbf{r}_d = [x_0, y_0, -R_Q]^T$
17 Emergency land	Emergency module	$z_d = -R_Q$

Table 2. Guidance commands and modules for each mode of the Autonomous Guidance System.

without error from the gyroscope outputs. Translational velocity $\mathbf{v}_Q$ is obtained from position via a low-pass filter.

State Feedback Module. A state feedback module with feedback linearisation [IVA15, Voo09, DSL09] stabilises the quadrotor and ensures accurate tracking of position and yaw commands. The position controller consists of cascaded state feedbacks, with the desired acceleration and velocity given by:

$$\ddot{\mathbf{r}}_d = K_{dr}\left(\dot{\mathbf{r}}_d - \dot{\mathbf{r}}\right) \quad \text{and} \quad \dot{\mathbf{r}}_d = K_{pr}\left(\mathbf{r}_d - \mathbf{r}_Q\right) \tag{3}$$

where $\mathbf{r}_d$ is the desired position and $\dot{\mathbf{r}}_d$ is subject to $\|\dot{\mathbf{r}}_d\| \leqslant v_{\max}$ where $v_{\max}$ is a velocity limit. Decoupling of the position and velocity feedbacks is required for the visual controller (see Eq. (6)). Linearising feedbacks then use the desired acceleration command $\ddot{\mathbf{r}}_d$ to determine the collective pseudo-input:

$$u_{col} = \frac{m_Q(g - \ddot{z}_d)}{K_T \cos\phi \cos\theta} \tag{4}$$

and the roll and pitch commands:

$$\phi_d = \arcsin\left(\frac{m_Q\left(\ddot{y}_d \cos\psi - \ddot{x}_d \sin\psi\right)}{K_T u_{col}}\right) \quad \text{and} \quad \theta_d = -\arcsin\left(\frac{m_Q\left(\ddot{x}_d \cos\psi + \ddot{y}_d \sin\psi\right)}{K_T u_{col}\cos\phi}\right)$$

subject to $\{|\phi_d|, |\theta_d|\} \leqslant a_{\max}$, where $a_{\max}$ is a maximum roll/pitch limit. The orientation controller is similarly defined by the state feedback and linearising feedbacks:

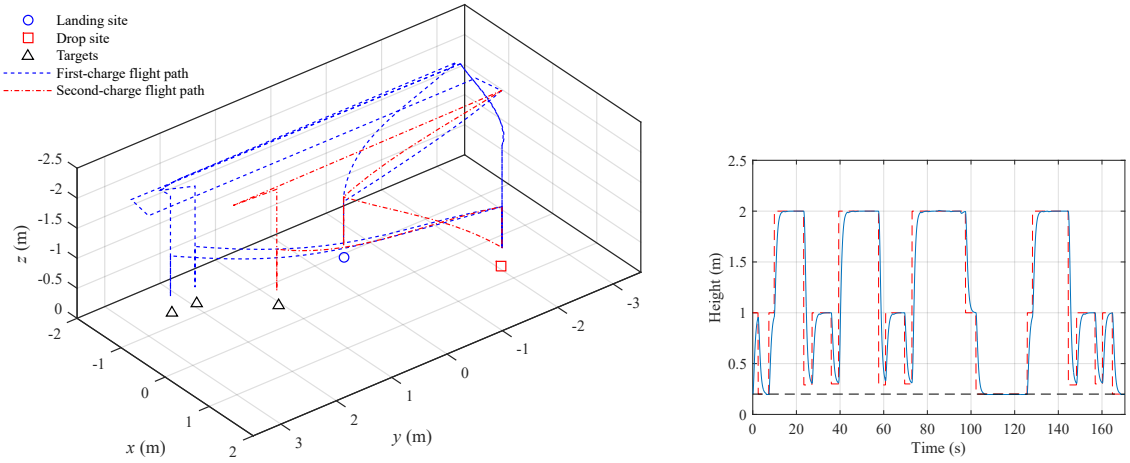
$$\dot{\boldsymbol{\omega}}_d = \mathbf{K}_{p\eta}\left(\boldsymbol{\eta}_d - \boldsymbol{\eta}_Q\right) - \mathbf{K}_{d\eta}\boldsymbol{\omega}_Q \quad \text{and} \quad \begin{bmatrix} u_{roll} \\ u_{pitch} \\ u_{yaw} \end{bmatrix} = \begin{bmatrix} \frac{I_x}{K_T L} & 0 & 0 \\ 0 & \frac{I_y}{K_T L} & 0 \\ 0 & 0 & \frac{I_z}{K_Q} \end{bmatrix} \dot{\boldsymbol{\omega}}_d$$

where $\mathbf{K}_{p\eta} = [K_{p\phi}, K_{p\theta}, K_{p\psi}]$ and $\mathbf{K}_{d\eta} = [K_{d\phi}, K_{d\theta}, K_{d\psi}]$. The four control variables u_{col}, u_{roll}, u_{pitch}, u_{yaw} are mixed to provide the true rotor speed inputs.

Search Pattern Module. The search pattern module defines a series of waypoints which the UAV follows after taking off. The waypoint locations are defined such that following each one from the beginning to the end of the search pattern results in near-exhaustive visual coverage of the environment floor. As each waypoint is reached a counter is incremented.

Emergency Module. The emergency module is similar to the state feedback module in structure. It is utilised only when an actuator fault occurs and an emergency landing is required. In this event, it is assumed that a single rotor has malfunctioned and the identity of the rotor is known. The opposing rotor is disabled and the thrust to the remaining two rotors increased to reduce the chance of a hard landing.

With one rotor inoperable, orientation control is neglected to avoid increasing the possibility of a crash.



(a) Flight path of the UAV (b) Height history of the UAV

Fig. 4. Results of a single simulation run.

In this case, the faulty motor and its opposite are deactivated, while the remaining two motors are controlled such that they each generate half of the thrust required to (as much as possible) gently land the quadrotor.

Object Tracking Module. The object tracking module is used to identify targets within the camera's field of view. The centroid of an identified object in the camera image provides coordinates which are utilised by the visual module.

Visual Module. The coordinates of the centroid are denoted $\mathbf{c}_{cntrd} = [c_x, c_y]$. The position module acts to drive the coordinates towards the centre of the camera image. To achieve consistency with the state feedback module, the error is defined as a function of the coordinates and the camera height, that is:

$$\mathbf{e} = \left| \frac{z_Q - z_{C/Q}}{f} \right| \begin{bmatrix} c_y \\ c_x \end{bmatrix} \tag{5}$$

where $z_{C/Q}$ is the camera position in $\mathcal{Q}$, and the centroid coordinates are swapped for consistency with the reference frame $\mathcal{Q}$. A proportional-integral controller centres the coordinates in the camera image and the target directly below the UAV and provides a desired velocity command to the state feedback module:

$$\begin{bmatrix} \dot{x}_d \\ \dot{y}_d \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \left(K_{pv}\mathbf{e} + K_{iv} \int \mathbf{e}\, dt \right) \tag{6}$$

where K_{pv}, K_{iv} are the proportional and integral gains, respectively. These commands override the horizontal velocity commands of the state feedback module (see Eq. (3)).

3.4. Running Simulations of the Model

Simulating a Single Run. Using MATLAB we ran a single simulation with the base site, drop site and target positions selected at random. In this instance, the mission was successful, i.e. all targets were deposited and the UAV returned to the landing site, and took 170s to complete. Fig. 4(a) presents the flight path followed by the UAV during the mission. It takes off and flies to the first waypoint at $[-1.5, -3]$ before following a path to the second waypoint at $[-1.5, 3]$. The red and blue targets are within both the camera field of view and the object detection algorithm's radius of interest while following this path. The UAV thus diverts to retrieve these targets. While the blue target is identified first, the forward motion of the vehicle results in the red target being closer to the UAV as it enters *Hover above target* mode. The red target is thus retrieved and deposited first. The UAV then returns to the first waypoint, follows the path to the second waypoint and the blue target is again identified. It is retrieved and deposited at the drop site. The UAV then returns to base and recharges the battery. The UAV then starts its second flight path returning to

Parameter	Probability
Mission success	0.8750
System fault in *Initialise* mode	0.0484
Actuator fault during mission	0.0335
Grasping fault during transportation	0.0105

Table 3. Results of Monte Carlo experiments.

Mode	Emergency land	Return to base
Idle	0.0	0.0
Take-off	0.0	0.0
Initialise	0.0	0.0
Search	0.0036	0.1460
Identify	0.0	0.0
Hover above target	0.0025	0.0539
Descend to grasp	0.0034	0.0156
Grasp	0.0	0.0002
Ascend	0.0017	0.0224
Transport	0.0010	0.0356
Descend to drop	0.0	0.0176
Drop	0.0	0.4698
Return to search	0.0	0.0051
Return to base	0.0003	–
Land	0.0012	–
Reacquire target	0.0	–
Emergency land	–	–

Table 4. Probabilities of transitioning to both *Emergency land* and *Return to base*.

the first waypoint and follows the path to the second. Finding no further targets, it proceeds to the third, fourth then fifth waypoints. While following the path to the sixth waypoint, it detects the green target, then retrieves and deposits it at the drop site. Having successfully deposited all three targets, the UAV returns to the landing site and the mission ends.

We can obtain information on the mission by analysing the height history of the UAV, shown in Fig. 4(b) (the dashed line indicates the height command). The UAV begins the mission by taking off to a height of one metre. It then immediately lands again, indicating that a system fault has occurred. The UAV then takes off again, ascends to the search height of two metres and continues the mission. We can see that the UAV descends multiple times during the mission to an altitude of around 0.3 m. This is the height at which it grasps and drops the targets which occurs six times. An additional descent occurs at around 100 s. This is followed by a period of around 20 s where the UAV rests on the floor. This is an instance of a low battery warning triggering a return to base.

Monte Carlo Experiments and Small Scale Simulation. For the Monte Carlo simulation, the full simulation is run for 2,000 iterations. The probability of mission success and of the different faults occurring are given in Table 3. The last three of these will be used to inform the first of our PRISM models described in Section 4. Reasons for mission failure include the following:

- At least one target is already in the drop zone as the mission begins. Since the UAV ignores objects in the radius around the drop site, it is unable to find this target and reaches the end of the search path having located the remaining targets.
- A target lands in the drop zone, but does not register as having been deposited. This occurs when the target is dropped during *Transport*, either due to a grabber fault or a low battery warning, and lands in the drop zone, rendering it invisible to the UAV.
- One or more targets are ignored by the UAV, despite being outside of the drop zone. This may occur when targets are located in the corners of the environment which are outside of the object tracking radius.
- An actuator fault occurs. This can happen at any time.

Transition probabilities are derived through multiple small scale simulations which focus on sections of the model only. Several modes may lead to both *Emergency land* and *Return to base*.

4. Abstract Markov Models

Inspired by the model described in Section 3, we describe a number of scenarios motivated by realistic situations for a range of autonomous vehicle applications, e.g. border patrol, exploration of unexplored terrain, and search and rescue operations. In each case we present a simplified scenario involving an *autonomous agent* searching for objects within a defined area.

For each scenario we describe abstract Markov models representing the choices controlling the search pattern of the agent and show how PRISM has been used for verification and controller synthesis.[2].

All of our models are finite state abstractions of a complex physical system. As explained in [FDW13], whereas the continuous dynamics of an autonomous system leads to a huge (possibly infinite) space of outcomes, the high-level decision making of an autonomous agent typically involves making choices amongst a small number of possibilities. In addition, decisions are made when thresholds determined by the system dynamics from multiple sensor readings are exceeded, rather than precise values returned by individual sensors. A notable abstraction in our models is the representation of the search space as a discrete grid with each grid cell corresponding to a cell of width and length 0.5 m both in the simulation model and the laboratory as described in Section 3 (height is not considered except in scenario 1). We also treat time as a discrete sequence of steps.

For all the presented PRISM models, the base, deposit site and battery capacity are model parameters. It is therefore straightforward to analyse models with different positions for the base, deposit site and battery capacity from those presented below by simply changing the corresponding parameter values. In our first scenario, our PRISM model closely mirrors the simulation model, in particular it follows the same underlying finite state machine (see Fig. 3). In addition, this PRISM model uses parameter values and probabilities derived from Monte Carlo experiments on the simulation model. Our subsequent scenarios were abstracted further, partially to overcome state-space explosion issues - a universal problem when using model checking [CKNZ11].

Scenario 1: Fixed controller. In [HIM+16] we introduced abstract MDP models representing the concrete simulation model of Section 3. The purpose was to investigate the viability of a framework for analysing autonomous systems using probabilistic model checking of an abstract model where quantitative data for abstract actions is derived from small-scale simulation models. Parameters for this model, including probabilities of different faults occurring were derived from Monte Carlo experiments (see Table 3).

The controller in this scenario is fixed and specifies that the agent searches the grid in a predetermined fashion, starting at the bottom left cell of the grid, travelling right along the bottom row to the bottom right cell, then left along the second row, and so on. The controller also specifies that if an object is found during search, then the agent attempts to pick up the object and, if successful, transports it to a specified deposit site. Whenever the agent's battery level falls below a specified threshold, it returns to the base to recharge and once the battery is charged continues the search. In both cases, search resumes from the previous cell visited, until all objects have been found or because the search cannot continue (e.g. due to an actuator fault or the mission time limit has been reached).

We used abstract MDP models and PRISM to analyse this scenario with a grid size of 7×4 and either 2 or 3 objects. Although the controller is fixed, nondeterminism is used to represent uncertainty in the environment, specifically the time taken for the agent to execute actions, bounds for which were obtained from our small-scale simulation models. The PRISM models contain modules for the agent's behaviour, movement, time and battery level, and objects. To reduce the size of the state-space, rather than encode the random placement of the objects within the model, we develop a model where objects have fixed coordinates and consider each possible placement of the objects. For example, in the case of two objects there are 378 different possible placements for the objects and each model with fixed placement has approximately 200,000 states. To obtain quantitative verification results for the model where objects are randomly placed we perform multiple verification runs by considering each possible placement of the objects and take an average.

We have analysed the following quantitative properties for this scenario.

1. *What is the minimum/maximum probability of mission success?* For this we use the PRISM specifications $\mathtt{P_{min=?}[F\ success]}$ and $\mathtt{P_{max=?}[F\ success]}$, where success is an atomic proposition labelling the states of the model for which the agent has returned to base having retrieved all of the objects.
2. *What is the minimum/maximum probability of a certain fault occurring?* For this we use the PRISM specifications $\mathtt{P_{min=?}[F\ fault]}$ and $\mathtt{P_{max=?}[F\ fault]}$, where fault is an atomic proposition labelling the states of the model for which the specified fault (either system, actuator or grasping fault) has occurred.
3. *What is the minimum/maximum probability of the mission completing by time T?* These properties

[2] The abstract MDP model and property files for each scenario are available from www.prismmodelchecker.org/subm/files/fac20/.

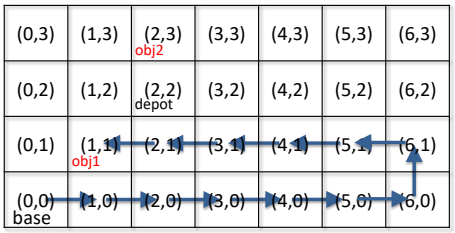


Fig. 5. Scenario 2: example positions for base, depot and objects (7×4 grid).

are specified in PRISM by $\mathtt{P}_{\mathtt{min}=?}[\,\mathtt{F}\ \mathsf{success} \wedge t \leqslant T\,]$ and $\mathtt{P}_{\mathtt{max}=?}[\,\mathtt{F}\ \mathsf{success} \wedge t \leqslant T\,]$ where t is the PRISM variable representing the elapsed time.

4. *What is the minimum/maximum probability of an actuator fault occurring?* These are specified in PRISM by $\mathtt{P}_{\mathtt{min}=?}[\,\mathtt{F}\ \mathsf{fail}\,]$ and $\mathtt{P}_{\mathtt{max}=?}[\,\mathtt{F}\ \mathsf{fail}\,]$, where the atomic proposition fail labels those states of the model for which an actuator fault has occurred.
5. *What is the minimum/maximum expected mission time?* To determine this, we first have to annotate our PRISM model with a reward structure, denoted *time*, that encodes elapsed time (see Definition 2.2), i.e. when the variable t is updated, once this reward structure has been added the properties can be specified in PRISM by $\mathtt{R}^{\mathtt{time}}_{\mathtt{min}=?}[\,\mathtt{F}\ \mathsf{done}\,]$ and $\mathtt{R}^{\mathtt{time}}_{\mathtt{max}=?}[\,\mathtt{F}\ \mathsf{done}\,]$, where the atomic proposition done labels those states for which the mission is complete (either with success or failure).

As the only form of nondeterminism in these models is from the environment (recall the controller is fixed), the minimum and maximum values provide lower and upper bounds on the actual results. In all cases, the corresponding results obtained via Monte Carlo simulation from the concrete simulation model (see Table 3) were comparable to our PRISM results (in that they were within the interval defined by our minimum and maximum values).

In the remainder of this section we synthesise optimal *controllers* for different scenarios with respect to the mission time. We achieve this using PRISM to encode the choices of the controller using nondeterminism. We remove the nondeterminism corresponding to environmental factors, e.g. the time taken to perform actions, as these are not choices of the controller. By moving to stochastic games [Sha53] we could separate the controller's choices from that of the environment. However, implementations of probabilistic model checking for such games, e.g. PRISM-games [KPW17], do not currently scale to the size of models we consider.

Scenario 2: Control of recharging. In this scenario we introduce choice as to when the battery is recharged. More precisely, recharging is no longer enforced when the battery reaches a predetermined lower threshold as in Scenario 1, but can be performed nondeterministically at any time during search. We assume that positions of the objects are fixed and the agent explores the grid in the predetermined fashion described for Scenario 1 above.

We use PRISM to find the minimum expected mission time and synthesise an optimal strategy that achieves this minimum for a suite of models involving two objects, varying the positions of the base, depot and objects. The synthesised strategies demonstrate that the optimal choice is to recharge when close to base, rather than waiting for the battery level to reach a threshold level.

As an illustrative example, consider the grid of size 7×4 in Fig. 5 where the objects are at positions $(1, 1)$ and $(2, 3)$, the base is at position $(0, 0)$ and the depot at $(2, 2)$. The agent first follows the search path shown in Fig. 5 and finds the first object. The controller then decides the agent should recharge by returning to base and then deposit the object at the depot. This differs from the controller in Scenario 1, as the battery level is not low at this point. By deciding to recharge at this point the agent on average only recharges once, whereas using the controller for Scenario 1 the agent on average recharges approximately twice.

The performance of the synthesised optimal controller is compared to that of the fixed controller used in Scenario 1 (which recharges when the battery level reaches a threshold) in Table 5. The results demonstrate that the synthesised controller offers a significant performance improvement over the controller of Scenario 1. The expected mission time drastically reduces, the probability of a successful mission increases and the expected number of battery recharges decreases.

Scenario 3: Control of search. We now generalise Scenario 2 to include control of the search path as well

Base	Depot	Object 1	Object 2	Expected mission time		Expected no. of battery charges		Probability of mission success	
				Scenario 1	optimal	Scenario 1	optimal	Scenario 1	optimal
(0,0)	(2,2)	(3,3)	(4,3)	285.8	138.0	2.301	2.021	0.949	0.975
(0,0)	(2,2)	(1,1)	(4,3)	252.2	147.3	2.181	1.002	0.956	0.975
(0,0)	(6,3)	(2,1)	(4,2)	292.4	178.0	2.364	1.056	0.948	0.970
(1,2)	(6,3)	(3,0)	(5,1)	212.4	83.93	1.015	0.203	0.962	0.987
(3,2)	(6,3)	(2,1)	(4,2)	218.9	117.3	1.127	1.032	0.960	0.980
(6,3)	(0,0)	(2,1)	(4,2)	221.4	119.0	1.090	1.021	0.960	0.978

Table 5. Scenario 2: performance of Scenario 1 controllers versus optimal strategies (7×4 grid).

```
// number of unexplored cells
formula n = gp0+gp1+gp2+gp3+gp4+gp5+gp6+gp7+gp8+gp9+gp10+gp11;

// probability of finding object in an unexplored cell
formula p = objs/n;
```

Fig. 6. PRISM code: probability of finding an object in an unexplored cell (12 cell grid).

as recharging. Since allowing freedom of movement increases the complexity of our model, we focus on the search mode of the agent and abstract other modes (including take-off, hover and grasp, see Section 3.3).

Having the positions of the objects as constants in the PRISM model is not feasible if our aim is to generate optimal and realistic controllers as this means that the agent knows the locations of the objects it is searching for. In such a situation, the optimal search strategy is clear: go directly to the objects and collect them. We initially considered using partially observable MDPs (POMDPs) and the associated extension of PRISM [NPZ17]. Using POMDPs we can hide the positions of the objects and synthesise an optimal controller, e.g. one that minimises the expected mission time. However, we found the prototype implementation did not scale as it implements only basic analysis techniques.

Subsequently we investigated modelling hidden objects with MDPs. This was found to be feasible by monitoring the unexplored cells and using the fact that the probability of an object being found in a cell that has not been explored is obj/n where obj is the number of objects still to be found and n the number of unexplored cells. In an $M\times N$ grid we associate the cell with coordinates (x, y) the integer (or *gridpoint*) $x+y\cdot M$. Before we introduce the PRISM code for an agent searching, we list the variables, formulae and constants used in the code:

- variable s is the state of the agent taking value 0 when searching and 1 when an object has been found;
- variables $posx$ and $posy$ are the current coordinates of the agent and formula gp returns the corresponding gridpoint;
- constants X and Y represent the grid size, where $X=M-1$ and $Y=N-1$;
- variable gpi for $0\leqslant i\leqslant(X+1)\times(Y+1)-1$ is 1 when cell with gridpoint i has not been visited, and 0 otherwise;
- variable $objs$ represents the number of objects yet to be found.

We assume the base and depot are fixed and located at position $(0, 0)$.

Figs. 6 and 7 give the PRISM code extracts relevant for finding an object for a grid with 12 cells when the agent is searching the cell with gridpoint 0. To search the cell the agent needs to be searching and located in the cell ($s=0$ and $gp=0$). If the cell has already been searched ($gp0=0$), then there is simply a nondeterministic choice as to which direction to move. If the cell has not been searched ($gp0=1$), then each choice includes the probability of finding an object using the formula in Fig. 6. The guard $p\leqslant 1$ prevents PRISM reporting modelling errors due to potentially negative probabilities. Boundaries are encoded in guards rather than using knowledge of the grid, e.g. it is not possible to move south or west in gridpoint 0, to allow automated model generation for different grids.

After an object has been found ($s=1$), the agent deposits it at the base and resumes search if there are more objects to find. Returning to base either to deposit or recharge is encoded by a single transition with time and battery consumption updated assuming the controller takes a shortest path to the base. This modelling choice is to reduce the state space. Also to reduce the state space, we add conditions to guards in the battery module to prevent the agent moving to a position from which it cannot reach base with the

```
// move east
[east] s=0 & gp=0 & gp0=0 & posx<X → (posx'=posx+1);
[east] s=0 & gp=0 & gp0=1 & posx<X & p⩽1 → p : (s'=1)&(posx'=posx+1)&(gp0'=0)
                                          + 1−p : (posx'=posx+1)&(gp0'=0);
// move west
[west] s=0 & gp=0 & gp0=0 & posx>0 → (posx'=posx−1);
[west] s=0 & gp=0 & gp0=1 & posx>0 & p⩽1 → p : (s'=1)&(posx'=posx−1)&(gp0'=0)
                                          + 1−p : (posx'=posx−1)&(gp0'=0);
// move north
[north] s=0 & gp=0 & gp0=0 & posy<Y → (posy'=posy+1);
[north] s=0 & gp=0 & gp0=1 & posy<Y & p⩽1 → p : (s'=1)&(posy'=posy+1)&(gp0'=0)
                                          + 1−p : (posy'=posy+1)&(gp0'=0);
// move south
[south] s=0 & gp=0 & gp0=0 & posy>0 → (posy'=posy−1);
[south] s=0 & gp=0 & gp0=1 & posy>0 & p⩽1 → p : (s'=1)&(posy'=posy−1)&(gp0'=0)
                                          + 1−p : (posy'=posy−1)&(gp0'=0);
```

Fig. 7. PRISM commands: searching cell with gridpoint 0.

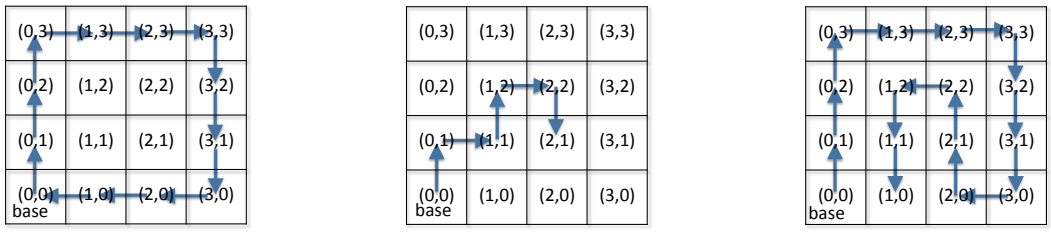
Grid size	No. of objects	Battery capacity	States	Transitions	Min expected mission time	Verification time (s)
3×3	1	16	3,478	7,778	14.22	0.036
3×3	2	16	6,598	14,913	24.44	0.073
4×4	1	24	403,298	1,016,387	24.00	1.248
4×4	2	24	860,689	2,212,391	38.60	2.840
5×4	1	28	5,332,892	13,942,821	29.20	14.94
5×4	2	28	11,841,031	31,526,709	46.27	38.77
6×4	1	32	64,541,199	172,990,992	34.33	197.4
6×4	2	32	149,723,921	408,008,297	53.94	5,140

Table 6. Scenario 3: model checking results.

remaining battery power. For example, for a 5×4 grid, two objects and a battery capacity of 28, together these modelling choices reduce the state space from 24,323,956 to 11,841,031.

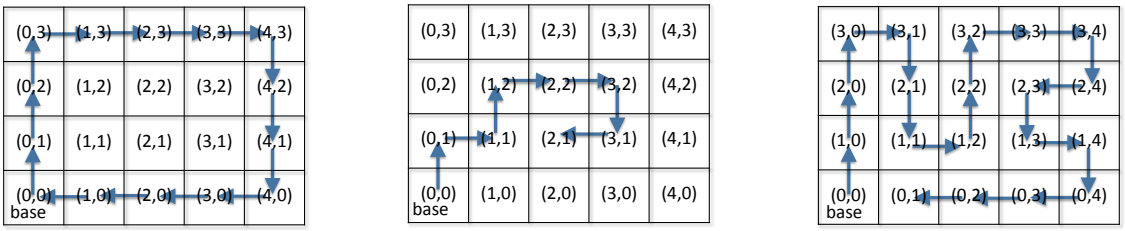
We synthesised optimal strategies for the minimum expected mission time for grids of varying sizes and number of objects. Table 6 presents model checking results in which we have chosen the minimum battery size that allows for a successful mission for the given grid. Figs. 8(a)–(b) and 9(a)–(b) present the optimal strategies when searching for a single object. The figures give the optimal search paths which require returning to base during the search to recharge the battery. By increasing the capacity of the battery, the optimal strategy does not require the battery to be recharged. Figs. 8(c) and 9(c) present optimal strategies for this situation. In each case, the time to return to base when the object is found must be taken into consideration as opposed to only the time it takes to search.

Scenario 4: Control of sensors. In this scenario we extend the power of the controller: as well as choosing the search path and when to recharge it can decide whether the search sensors are in a low or high power mode. In the high power mode the agent can search a cell, while in the low power mode it is only possible to



(a) path 1 and recharge (b) path 2 after recharge (c) path (no recharge)

Fig. 8. Scenario 3: optimal controllers for 4×4 grid, battery capacities 24 and 28.



(a) path 1 and recharge (b) path 2 after recharge (c) path (no recharge)

Fig. 9. Scenario 3: optimal controllers for 5×4 grid, battery capacities 28 and 32.

```
// sensors in high power mode and cell unexplored
[east0] s=0 & c=1 & gp=0 & gp0=1 & posx<X &p≤1 →
    p : (s'=1)&(posx'=posx+1)&(gp0'=0)&(c'=0) + 1−p : (posx'=posx+1)&(gp0'=0)&(c'=0);
[east1] s=0 & c=1 & gp=0 & gp0=1 & posx<X &p≤1 →
    p : (s'=1)&(posx'=posx+1)&(gp0'=0)&(c'=1) + 1−p : (posx'=posx+1)&(gp0'=0)&(c'=1);

// sensors in lower power mode and cell unexplored
[east0] s=0 & c=0 & gp=0 & gp0=1 & posx<X → (posx'=posx+1)&(c'=0);
[east1] s=0 & c=0 & gp=0 & gp0=1 & posx<X → (posx'=posx+1)&(c'=1);

// cell already explored (does not matter the sensors power mode)
[east0] s=0 & gp=0 & gp0=0 & posx<X → (posx'=posx+1) & (c'=0);
[east1] s=0 & gp=0 & gp0=0 & posx<X → (posx'=posx+1) & (c'=1);
```

Fig. 10. PRISM commands: searching cell 0, moving east and switching sensors off/on.

traverse the cell. The high power mode for search is expensive in terms of time and battery use and can be unnecessary, e.g. when travelling over previously explored cells or returning to base to deposit or recharge. Again we assume the base and depot are fixed and located at position $(0, 0)$. The PRISM model for this scenario extends that for Scenario 3 as follows. A variable c is added to the agent module, taking value 0 and 1 when its sensors are in low and high power modes respectively. The (nondeterministic) choices of the controller are then extended such that when deciding the direction of movement it also decides the power mode of the sensors for traversing the next cell. To aid analysis of the synthesised strategies, the action labels for direction of search include the power mode of the sensors, e.g. *south1* corresponds to moving south and selecting high power mode and *west0* to moving west and selecting low power mode.

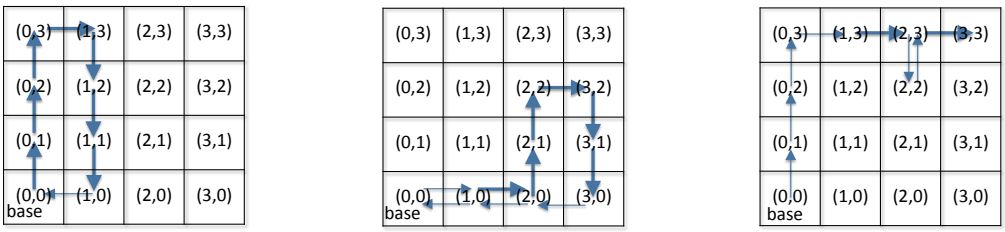
The PRISM code extract in Fig. 10 gives commands for moving east from cell with gridpoint 0 based on those in Fig. 7 for Scenario 3. The first two commands consider the case where the agent's sensors are in high power mode (c=1) and the cell is unexplored. In both cases, since the sensors are in high power mode and the cell is unexplored, the probability of finding an object is as for Scenario 3. The difference is that in the first command the sensors are switched to lower power mode, while in the second the sensors remain in high power mode. The third and fourth commands represent the case when the sensors are in lower power mode and the cell is unexplored. Since the sensors are in lower power mode, the cell remains unexplored and there is no chance of finding the object. The final two commands consider the case where the cell has been previously explored. The PRISM model is also updated so that the time passage and battery consumption reflect the sensor's current power mode.

Table 7 presents model checking results for this scenario including both those for the battery capacity from Scenario 3 (see Table 6) and for the minimum battery capacity required for a successful mission. Comparing with Table 6, allowing low and high power modes reduces the mission time, allows the mission to be completed with a smaller battery capacity and reduces recharging.

Comparing optimal strategies for Scenario 3 in Figs. 8 and 9 and those for Scenario 4 with the same battery capacity, the only difference is that the low power mode is used when revisiting a cell. In Fig. 11 we present an optimal strategy for a 4×4 grid and battery capacity of 18. In this case, it is not feasible to complete the mission without using the lower power mode. Smaller arrows represent when the sensors are

Grid size	No. of objects	Battery capacity	States	Transitions	Min expected mission time	Verification time (s)
3×3	1	12	53,367	222,557	12.78	0.206
3×3	1	16	80,107	351,209	12.11	0.272
3×3	2	12	103,063	435,302	19.83	0.45
3×3	2	16	154,911	687,246	18.88	0.420
4×4	1	18	18,445,790	90,303,355	21.88	58.58
4×4	1	24	27,587,864	139,945,165	20.50	83.60
4×4	2	18	36,379,747	180,055,826	32.22	124.8
4×4	2	24	54,464,317	279,117,740	30.60	181.9

Table 7. Scenario 4: model checking results.



(a) first path (b) second path (c) third path

Fig. 11. Scenario 4: optimal controller for 4×4 grid, battery capacity 18 and one object.

in lower power mode. The move south during the third path before searching the final cell $(3,3)$ might not appear optimal. However, immediately before this step there is an equal chance of finding the object in the two remaining unexplored cells $(2,3)$ and $(3,3)$. By moving south after searching $(2,3)$ the time of returning to base is reduced when the object is found, at the cost of increasing the time to reach and search $(3,3)$ when the object is not found. In fact it is the case that initially moving east from $(2,3)$ also yields an optimal strategy, but was not the strategy synthesised by PRISM.

We also implemented the fixed strategy for the low power mode which uses this mode when either revisiting a gridpoint, i.e. one that has already been searched, or returning to base. We find that this strategy yields the same results as those presented in Table 7.

Scenario 5: Control of multiple agents. We now consider the case where there are multiple agents working together to find a single object. We extend the PRISM model for Scenario 4 by having modules for two agents. In addition, since more than one cell can be explored at the same time, to simplify the PRISM code each cell is modelled as a separate module. The probability of finding an object is now dependent on both agents, and therefore we model this in a separate module, using variables *u1* and *u2* to indicate if the first or second agent finds the object respectively.

The agent modules are presented in Fig. 12. Variables *pos1x* and *pos1y* represent the position of the first agent and *pos2x* and *pos2y* the second. Constants *basex* and *basey* give the position of the base and formula *base* the corresponding gridpoint. The search commands from the previous scenarios are modified and now synchronise on the action *search* with the gridpoint modules. Each command checks the variables *u1* and *u2* which indicate if an object has been found (see Fig. 12), since once the object is found, the agents return to base as the mission is complete. As the direction of movement is not encoded in the action *search*, preventing an agent moving in directions from which it cannot return to base with its remaining battery power is now encoded in formulae *move_east*, *move_west*, *move_north* and *move_south*.

For each cell in the grid there is a corresponding gridpoint module. The gridpoint modules for a 3×3 grid are presented in Fig. 13. By using the constants *ki* we only need to explicitly construct the first gridpoint module and then use renaming. In the module for the first gridpoint (see Fig. 13), variable *gp0* is 1 when the cell is unexplored and 0 otherwise.

As stated above the probability of an agent finding an object is now a separate module, presented in Fig. 14. As before, the probability of an unexplored cell containing an object is $1/n$ where n is the number of unexplored cells (see Fig. 6). Formulae *s1* and *s2* evaluate to 1 if *agent1* and *agent2* are searching unexplored cells respectively. If the agents are searching different unexplored cells, then each agent has a chance of finding the object, but both cannot find the object as it cannot be in two places at once.

Table 8 presents model checking results for Scenario 5. As expected we see that searching with two agents

```
// module for agent1
module agent1

  pos1x : [0..X] init basex; // x coordinate of agent1
  pos1y : [0..Y] init basey; // y coordinate of agent1

  // search (remain on grid and have sufficient battery)
  [search] u1=0 & u2=0 & pos1x<X & move_east -> (pos1x'=pos1x+1); // east
  [search] u1=0 & u2=0 & pos1x>0 & move_west -> (pos1x'=pos1x-1); // west
  [search] u1=0 & u2=0 & pos1y<Y & move_north -> (pos1y'=pos1y+1); // north
  [search] u1=0 & u2=0 & pos1y>0 & move_south -> (pos1y'=pos1y-1); // south

  // found object and not at base (go back to base)
  [end] (u1=1|u2=1) & !(agent1=base) -> (pos1x'=basex) & (pos1y'=basey);

  // mission complete
  [end] (u1=1|u2=1) & agent1=base -> true;

endmodule

// agent2 (rename agent1)
module agent2 = agent1[pos1x=pos2x, pos1y=pos2y] endmodule
```

Fig. 12. PRISM code: modules for *agent1* and *agent2* of Scenario 5.

```
// constants used for renaming gridpoints
const int k0 = 0;
          ⋮
const int k8 = 8;

// module for gridpoint 0
module gridpoint0

  gp0 : [0..1] init 1; // status of gridpoint0 (0 - explored and 1 - unexplored)

  // one of the agents searches the cell
  [search] (agent1=k0|agent2=k0) & (gp0=1) -> (gp0'=0);

  // cell already searched or not being searched
  [search] !((agent1=k0|agent2=k0) & (gp0=1)) -> true;

endmodule

// construct further gridpoints by renaming gridpoint0
module gridpoint1 = gridpoint0[gp0=gp1, k0=k1] endmodule
                  ⋮
module gridpoint8 = gridpoint0[gp0=gp8, k0=k8] endmodule
```

Fig. 13. PRISM code: module for gridpoints of Scenario 5 (3×3 grid).

```
// current gridpoint of agent1 and agent2 (derived from coordinates)
formula agent1 = pos1x+pos1y*(X+1);
formula agent2 = pos2x+pos2y*(X+1);

// retrieval probabilities module
module probabilities

   u1 : [0..1] init 0; // agent1 finds the object
   u2 : [0..1] init 0; // agent2 finds the object

   // agent1 and agent2 are searching different unexplored cells
   [search] s1=1 & s2=1 & agent1!=agent2 & 2*p<=1 -> p : (u1'=1)
                                                   + p : (u2'=1) + 1-2*p : true;

   // agent1 and agent2 searching the same unexplored cell
   // suppose each has the same chance of finding the object
   [search] s1=1 & s2=1 & agent1=agent2 & p<=1 -> p/2 : (u1'=1)
                                                + p/2 : (u2'=1) + 1-p : true;

   // agent1 is searching unexplored cell while agent2 is not
   [search] s1=1 & s2=0 -> p : (u1'=1) + 1-p : true;

   // agent2 is searching unexplored cell while agent1 is not
   [search] s1=0 & s2=1 -> p : (u2'=1) + 1-p : true;

   // neither agent searching an unexplored cell
   [search] s1=0 & s2=0 -> true;

endmodule
```

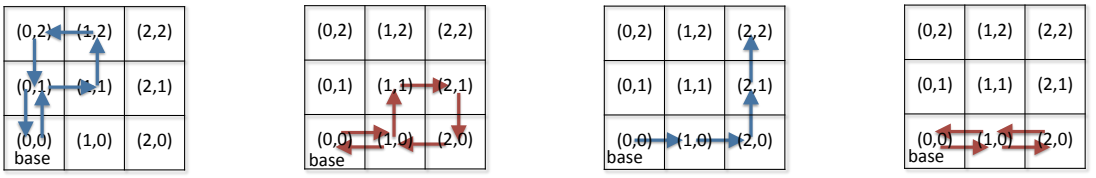
Fig. 14. PRISM code: retrieval probabilities module of Scenario 5.

Grid size	Battery capacity	States	Transitions	Min expected mission time	Verification time (s)
3×3	16	53,832	249,588	12.00	0.285
3×3	20	102,857	508,446	11.05	0.404
4×4	24	15,555,103	91,859,618	17.04	520.4
4×4	28	28,281,179	175,613,268	17.04	82.23
5×4	28	293,118,691	1,861,895,602	20.40	1,069
5×4	32	525,432,653	3,482,656,934	20.40	4,855

Table 8. Scenarios 5: model checking results.

can reduce the mission time over a single agent (see Table 6). Figs. 15 and 16 present optimal strategies for a grid of size 3×3 when the battery capacity is 16 and 20 respectively, and Fig. 15 for a grid of size 4×4 and a battery capacity of 24. The optimal strategies are represented by the paths of the two agents before the object is found. As for the previous scenarios, as soon as the object is found the agents return directly to base. Neither the second path of *agent2* in Fig. 15 nor the second path of *agent1* in Fig. 17 contribute to the search. In both situations after recharging, there is only one cell to search ($(2,2)$ and $(3,3)$ respectively) and there is no gain in sending more than one of the agents to search this cell. Although in Fig. 15, *agent1* covers all cells, *agent2* covers the cells it searches in Fig. 15(b) at an earlier point in time, therefore reducing the expected mission time.

In all cases presented in Table 8 it is feasible for the agents to search the grid without recharging their batteries. However, this is not always optimal due to the time required to return to base after finding the object. For example, we can see this in Figs. 15 and 17 where the battery is recharged before finishing search. For the case of a 3×3 grid, increasing the battery capacity yields an optimal strategy that does not need to



(a) path 1 (*agent1*) (b) path 1 (*agent2*) (c) path 2 (*agent1*) (d) path 2 (*agent2*)

Fig. 15. Scenario 5: optimal controller for 3×3 grid, battery capacity 16 and one object.

(a) path 1 (*agent1*) (b) path 1 (*agent2*)

Fig. 16. Scenario 5: optimal controller for 3×3 grid, battery capacity 20 and one object.

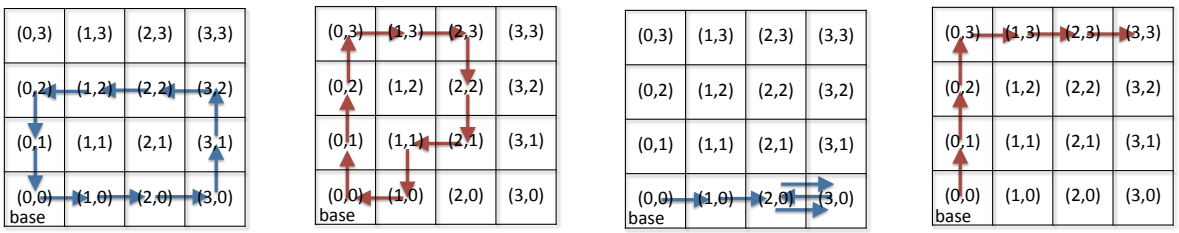
recharge, as shown in Fig. 16. On the other hand, as demonstrated in Table 8, for the remaining grid sizes considered, increasing the battery capacity does not change the optimal strategy.

Scenario 6: Control of multiple agents with idle mode. As discussed for Scenario 5, in certain situations there is no gain in both agents searching. For this reason in this scenario we add the ability for the controller to search using only one agent while the other idles at the base. Although this cannot reduce the mission time it can reduce power consumption and wear and tear.

Idling is introduced to the abstract MDP model through additional variables and the reward structure for time passage is updated to reduce the reward gained when an agent idles[3]. The optimal strategy will then choose idling over unnecessary movement, however as the reward is not reduced significantly it will use both agents to search when this can save time.

Table 9 presents model checking results for Scenario 6. The reduced mission time from Scenario 5 to 6 is due to the change made to the reward structure and the generated optimal strategies yield the same expected mission time as those synthesised for Scenario 5. Fig. 18 presents optimal strategies for a grid of size 3×3. This strategy is very different from that for Scenario 5 as in this case *agent1* searches the majority of the grid, while *agent2* searches only a small portion and returns to base and idles while *agent1* completes its search. For the 4×4 grid the optimal strategy is initially the same as for Scenario 5 (see Fig. 17). However, in the second phase of the search there is no path for *agent1*, instead it idles at base while *agent2* searches the remaining cell.

[3] See www.prismmodelchecker.org/subm/files/fac20/



(a) path 1 (*agent1*) (b) path 1 (*agent2*) (c) path 2 (*agent1*) (d) path 2 (*agent2*)

Fig. 17. Scenario 5: optimal controller for 4×4 grid, battery capacity 24 and one object.

Grid size	Battery capacity	States	Transitions	Min expected mission time	Verification time (s)
3×3	16	153,063	910,392	11.93	0.701
3×3	20	279,588	1,715,934	1104	1.724
4×4	24	38,165,612	255,234,876	17.46	148.0
4×4	32	101,827,888	719,801,776	17.25	412.2
5×4	28	691,136,157	4,826,058,012	20.4	14,671
5×4	32	1,221,178,098	8,802,459,931	20.4	38,045

Table 9. Scenarios 6: model checking results.

(a) path (*agent1*) (b) path (*agent2*)

Fig. 18. Scenario 6: optimal strategy for 3×3 grid, battery capacity 16 and one object.

5. Returning to the Concrete Simulation Model

In this section we address the following questions: *how can our synthesised optimal strategies be deployed as controllers for autonomous systems? are the resulting controllers effective?* and *how might our approach be applied to a larger search area?*

To answer these questions, we first describe how we should adapt our concrete simulation model in order to implement the synthesised strategies of Section 4. Then, we show in detail how this approach is applied for one of the scenarios, namely Scenario 3. Next, we compare the performance of the controller using the simple search strategy (*pattern search*) described in Section 3 with the controller adopting the search pattern (*smart search*) of the synthesised optimal strategies for Scenario 3. Finally, we show how a symmetry-based approach can be used to explore a larger search area by recycling strategies generated for a smaller area, presenting results obtained by implementing the approach on a grid of size 10×10 – again for Scenario 3.[4]

Deploying the Synthesised Optimal Strategies - The General Approach. For a given concrete model, for the approach to be applicable, one must construct an abstract MDP model such that one can map any reachable concrete state (i.e. values of the variables of the concrete system) from which we require a decision from the generated optimal strategy to a reachable abstract state (i.e. values of the variables of the abstract MDP model) which is an abstraction of the concrete state.

For example, in the case of Scenario 3, where the concrete states from which we require such a decision are those in which the agent is in search mode and it has to decide where to search next or whether to return to base and recharge the battery, the map is defined as follows:

- all such states are mapped to states for which the abstract MDP model is in search mode (s=2);
- the real-valued coordinates of the agent in the concrete system are mapped to the integer-valued coordinates of the grid point the agent is in (*posx* and *posy*);
- the values of variables in the concrete model encoding the grid points that have been searched are mapped to equivalent values of variables in the abstract MDP model (*gpi* for $0 \leqslant i \leqslant 24$);
- the number of objects found in the concrete model is mapped directly to the number of objects found in the abstract MDP model (*obj*);
- the real value representing the current battery level is mapped to an integer value representing the battery's level in the abstract MDP model (b).

For the scenarios we have considered where there is more than one agent, we would need to map the individual agents' real-valued coordinates and battery levels to grid points and integer values of the individual abstract

[4] All of our PRISM and MATLAB files are available from www.prismmodelchecker.org/subm/files/fac20/.

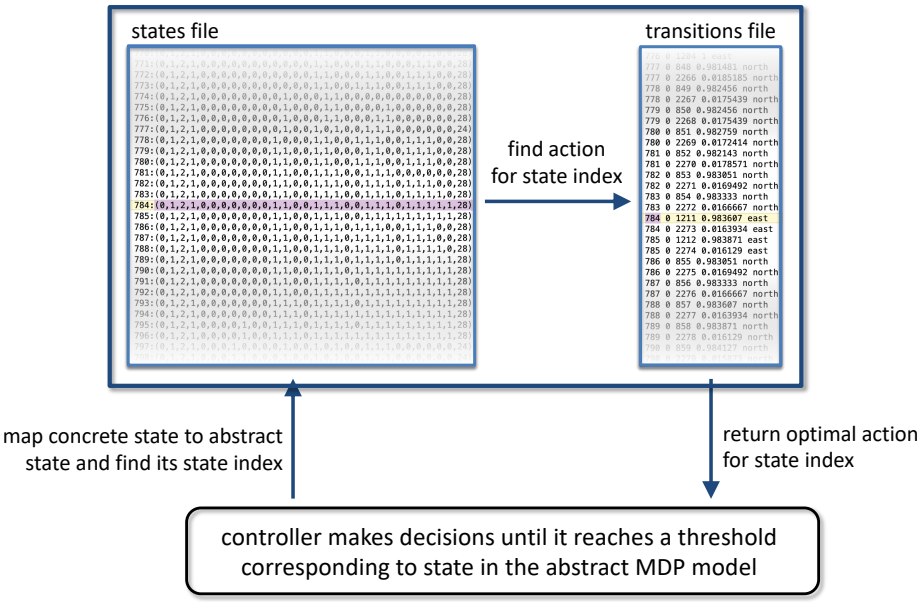


Fig. 19. Deployment of an optimal strategy of the abstract MDP model on a concrete system.

agents, and when additional modes are included (either low power or idle modes) we would need to include these in the mapping.

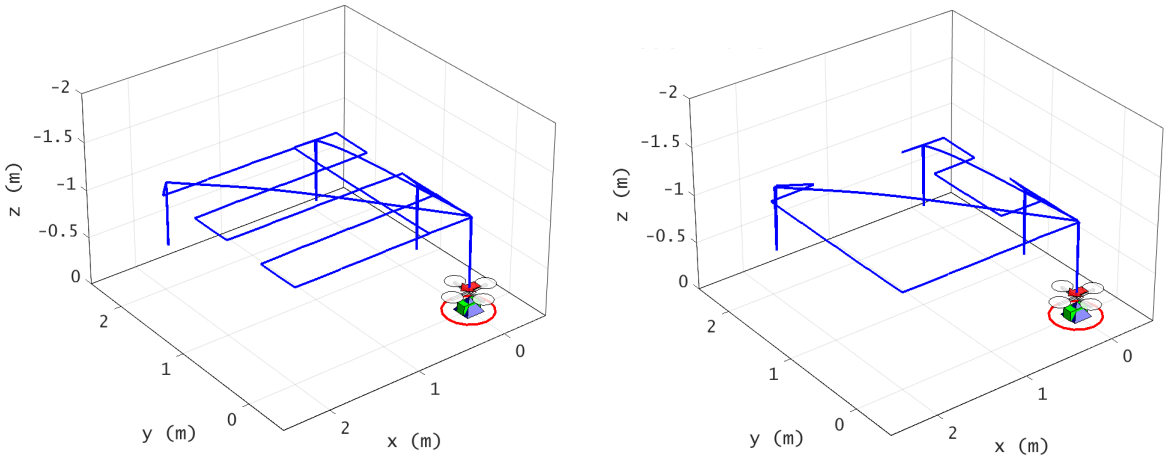
Fig. 19 demonstrates how the approach can be deployed in a concrete system when this mapping is available. The controller has access to two files generated by PRISM during optimal strategy synthesis. The first file lists the states of the PRISM model and their indices, and the second lists the optimal transition in each state index together with the associated action and the state indices and probabilities resulting from performing the transition. When a decision is required from a concrete state the index of the corresponding abstract state is identified which allows for the identification of the optimal action to perform.

Deploying the Synthesised Optimal Strategies - An Example (Scenario 3). Since the concrete simulation model of Section 3 is built in a modular way, it is straightforward to toggle between the search strategies *pattern search* and *smart search*. Recall that the search strategy *pattern search* follows a sequence of waypoints, returning to base if an object is found, the end of the search path is reached, or a low battery warning is received.

On the other hand, for *smart search* the controller refers to the synthesised strategy for Scenario 3 to decide upon both the search path and when to recharge the battery. To implement this, the controller has access to the state and transition files generated by PRISM during optimal strategy synthesis which, for convenience, we will refer to as `states_N` and `transitions_N` respectively (where `N` indicates the number of objects in the grid).

In order for the concrete simulation model and abstract MDP models to be consistent, we updated the abstract MDP model to include the possibility that some of the objects are not found during search and to suppose there are three objects to be found. In addition, in the abstract MDP model for Scenario 3 the agent searches a grid point and moves to the next grid point in one transition (see Fig. 7) which are separate steps in the concrete simulation model. Therefore, again for consistency, the abstract MDP model was updated so that the agent only moves to the next grid point when it does not find an object.

We also required some preprocessing of the files generated by PRISM to make the approach feasible. In particular, during strategy synthesis PRISM lists the optimal transitions in *all* states of the model, while the lists `states_N` and `transitions_N` only contain states that are reachable when following the synthesised strategy. For the presented scenario, this reduces the `states_N` list from approximating $400,000,000$ entries to approximately $4,000$ entries. This however does come at a cost, since the abstract MDP model does not include the stochastic behaviour of Section 3.2, after the preprocessing in the concrete simulation model it is possible to end up in a state for which the optimal controller does not know what to do. We present



(a) *pattern search* (mission time: 171.90 seconds) (b) *smart search* (mission time: 130.50 seconds)

Fig. 20. Flight path of a single simulation run under the different search strategies.

two simple solutions to this: a purely *smart search* implementation in which, if such a state is reached, we assume the mission has failed; and a *hybrid search* implementation in which we revert to the *pattern search* implementation when such a state is reached. More effective approaches would be possible, but these proved to be sufficient for our initial investigation.

Recall from Section 4, a state of the abstract MDP model is a tuple containing integer variables s (the state of the agent), *posx* and *posy* (the current coordinates of the agent), *obs* (the number of objects still to be found), b (the current battery level of the agent) and boolean variables *gpi* for each gridpoint (indicating whether the gridpoint has been explored). For a 4×4 grid each entry in `states_N` has the following form, where *ind* denotes the state index:

$$ind{:}(s, posx, posy, objs, gp0, gp1, gp2, gp3, gp4, gp5, gp6, gp7, gp8, gp9, gp10, gp11, gp12, gp13, gp14, gp15, b)$$

The list `transitions_N` contains entries of the form:

$$ind_i \;\; choice \;\; ind_j \;\; p \;\; act$$

where ind_i and ind_j are the indices of the source and target states of the transition, *choice* is an integer index of the transition, p is the probability of the transition and *act* is an optional action label.

When searching under the *smart search* strategy and deciding where to search next, the controller finds its current state index (*ind*) by accessing the list `states_N` and consults `transitions_N` to determine the correct action to take (see Fig. 19). In the concrete simulation model the objects are fixed, and so the probabilities listed in the transition file are ignored (the only probabilities in Scenario 3 correspond to the chances that an object is found at a grid point). Fig. 20 presents the flight path of a single simulation run both for *pattern search* and *smart search*.

Monte Carlo Simulation. We ran 4,000 simulations for *pattern search* and *hybrid search* on a grid of size 5×5 with 3 objects with both the base and deposit site fixed at $(0, 0)$. In 332 instances of the 4,000 simulations with *hybrid search*, a rate of 8.3%, there was a switch from *smart search* to *pattern search*. In order to gather *smart search* statistics these 332 instances were reclassified as failed missions where the quad was unable to find the object.

Table 10 presents the performance metrics and fault rates obtained from the simulations and Fig. 21 the distribution of the mission times for *pattern search* and *hybrid search*. As can be seen using either *smart search* or *hybrid search* yields a substantial improvement in both mission time and battery usage over *pattern search*. One example of the possible gains can be seen in Fig. 20 where the mission time is reduced by approximately a quarter. This decrease in mission time also yields a decrease in the chance of an actuator fault occurring, as the UAV is active for less time. However, there is a cost as the number of mission failures increases. This is to be expected in both cases. For *smart search* we have additional mission failures when we reach a state in which the optimal controller does not know what to do. For *hybrid search*

		Pattern	Smart	Hybrid	Hybrid (and switch)
Metrics	Number of Simulations	4000	4000	4000	332
	Success Rate	69.7%	63.3%	63.3%	47.0%
	Avg. Mission Time(s)	178.4	139.1	139.1	246.7
	Avg. Battery Usage	100.1	75.28	75.30	138.9
Fault Rates	System Initialisation	0.08%	0.03%	0.03%	0.03%
	Actuator	3.33%	2.65%	2.88%	2.71%
	Mission Time Exceeded	2.05%	0.00%	1.00%	12.1%
	Missed Objects	24.85%	34.0%	28.9%	38.3%

Table 10. Performance metrics and fault rates for the different search strategies.

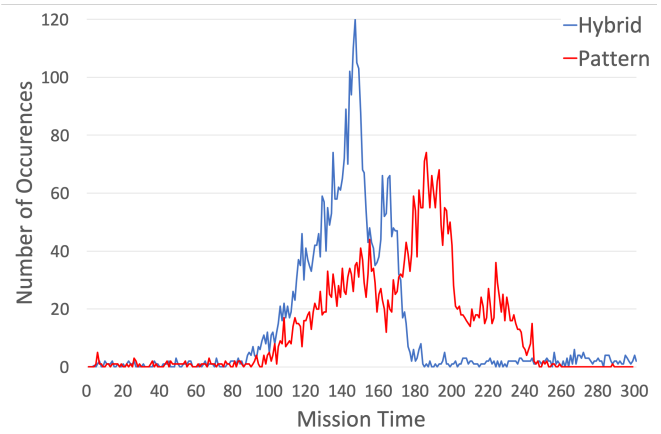


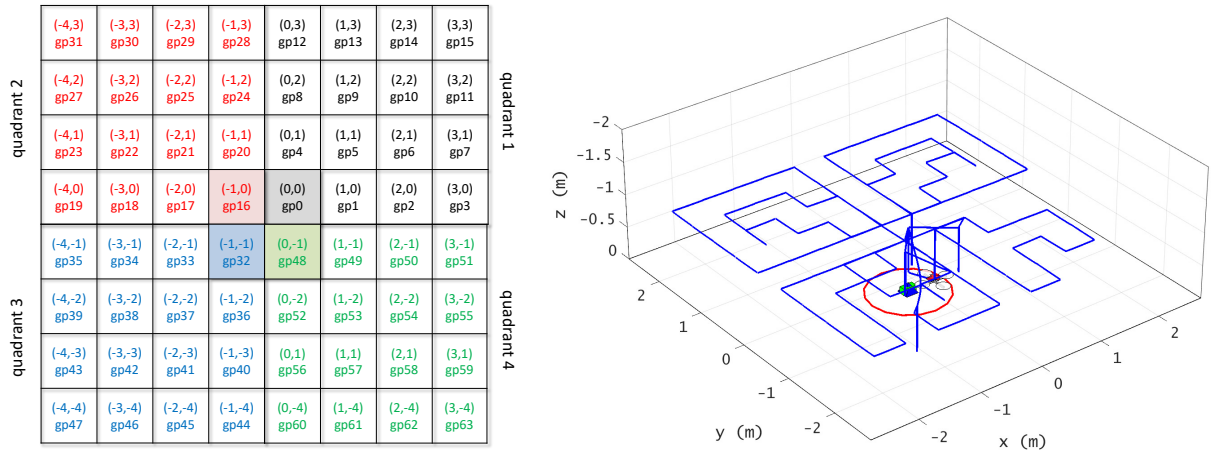
Fig. 21. The distribution of mission times for *pattern search* and *hybrid search*.

after switching strategies there is a chance that certain areas of the grid are left unexplored, and therefore objects are missed during the mission. In addition, after switching there is a possibility that areas that have previously been searched are searched again, which increases both the mission time and the likelihood that we run out of time, and consequently do not successfully complete the mission. This can be seen in the final column of Table 10 which gives the statistics for the simulation runs which switch from *smart search* to *pattern search* and in Fig. 21 where *hybrid search* has a longer tail than *pattern search*. The lower success rate when switching is also to be expected as the switch in search strategy is caused by some error or fault occurring. These limitations of *hybrid search* could certainly be mitigated by implementing a more robust switching policy that takes better account of what areas have been searched, which would reduce both the mission time and the chance of missing an object.

Extending the Approach to a Larger Search Area. As demonstrated by the results in Table 6, the presented approach will not scale to large search areas. There is certainly scope for extending the applicability of our approach by using model reduction techniques such as bounded model checking [KPR16], symmetry reduction [MDC06, DMP09, GL19] or abstraction-refinement [KKNP10]. These techniques usually involve merging states or abstracting parts of the underlying system; or require the existence of multiple interacting identical components.

We outline an approach for extending the search area using a simple application of symmetry. By dividing the search area into smaller regions, we can synthesise strategies by exploiting those created for the smaller regions. We describe the approach for a grid of size 8×8, which is illustrated in Fig. 22(a). Let us assume that there are N objects placed within the grid.

We first synthesise strategies for a grid of size 4×4 for the following scenarios: search region $1\leqslant i\leqslant 4$ and $1\leqslant k\leqslant N$ objects to find (when $i{=}1$ we need only consider the case when $k{=}N$ since we are searching the first region). The UAV search starts in gridpoint 0 (*gp0*) with coordinates $(0,0)$ shown in the bottom left corner of the first quadrant in Fig. 22(a) (the black grid) and follows the optimal synthesised strategy generated for $i{=}1$ and $k{=}N$. If all N objects are found in the first quadrant, search is complete. However, if $0{<}l\leqslant N$ objects remain to be found, the UAV moves to gridpoint *gp16* in the bottom right of quadrant 2 in Fig. 22(a) (the red grid) with coordinates $(-1,0)$ and follows a modified version of the synthesised strategy for $i{=}2$



(a) Expanded search space, 8×8 grid

(b) Flight path of a single simulation run, 10×10 grid.

Fig. 22. Expanding the search area using symmetry.

and $k{=}l$. The values of the variables and transitions in the generated states and transitions lists `states_1` and `transitions_1` are transformed under the mappings:

1. $(s, posx, posy, obs, gp0, gp17, \ldots, gp15, b) \mapsto (s, -(posx{+}1), posy, obs, gp16, gp17, \ldots, gp31, b)$;
2. $east \mapsto west$, $west \mapsto east$ and $act \mapsto act$ for $act \in \{north, south\}$.

If, after exploring the second quadrant, there are still $0{<}m{\leqslant}N$ objects to be found the UAV moves to gridpoint $gp32$ in the top right of quadrant 3 in Fig. 22(a) (the blue grid) with coordinates $(-1,-1)$ and follows the strategy derived from that synthesised for the case when $i{=}3$ and $k{=}m$. This time the values of the variables and transitions in the generated states and transition lists are transformed under the mappings:

1. $(s, posx, posy, obs, gp0, gp17, \ldots, gp15, b) \mapsto (s, -(posx{+}1), -(posy{+}1), obs, gp32, gp33, \ldots, gp48, b)$;
2. $east \mapsto west$, $west \mapsto east$, $north \mapsto south$ and $south \mapsto north$.

The search then continues to quadrant 4 if necessary in a similar manner. An example of a simple run of this search strategy over an expanded space of size 10×10 is given in Fig. 22(b).

We ran 4,000 simulations for a *pattern search* and *hybrid search* where each quadrant is of size 5×5, there are 3 objects, both the base and deposit site fixed in the centre and the agent has a battery capacity of 34. The *pattern search* takes the form of a simple spiral: first searching the outer border and then working inwards. This was chosen over an outward spiral as it would involve less time traversing previously searched gridpoints. Table 11 presents the performance metrics and fault rates from the simulations and the distributions of the mission times is given in Fig. 23. Recall that in the *smart search* implementation we assume the mission has failed if we end up in a state for which the optimal controller does not know what to do, while the *hybrid search* implementation reverts to the *pattern search* implementation when such a state is reached. As the results demonstrate we again see that our approach can yield significant improvements in both the mission time and battery usage. On the other hand, there is a slight increase in the chance of missing at least one of the objects, however this is not significant (for *smart search* the chance of missing objects is expected to be higher due to the increase in failed missions).

6. Conclusions

We have described a concrete simulation model for the simulation of a quadrotor UAV in operation inside a small, constrained environment. We then developed abstract MDP models for a sequence of scenarios relevant to UAVs and other autonomous agents. These scenarios incorporate nondeterministic choice for a range of aspects including search pattern, battery recharging, sensor control and multiple agent cooperation. Next, we described how our optimal strategies generated by PRISM can be deployed within our concrete

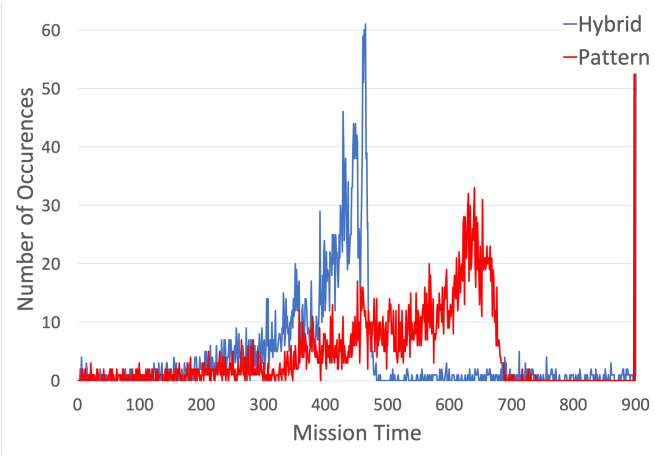


Fig. 23. The distribution of mission times for *pattern search* and *hybrid search* for the expanded search space.

		Pattern	Smart	Hybrid	Hybrid (and switch)
Metrics	Number of Simulations	4000	4000	4000	403
	Success Rate	63.5%	58.2%	58.3%	40.2%
	Avg. Mission Time(s)	531.6	380.8	381.1	750.0
	Avg. Battery Usage	311.5	209.4	209.6	443.9
Fault Rates	System Initialisation	0.03%	0.08%	0.08%	0.08%
	Actuator	8.6%	6.3%	7.1%	8.0%
	Missed Objects	26.1%	35.5%	27.2%	17.9%

Table 11. Performance metrics and fault rates for the expanded search space.

model, and demonstrated the effectiveness of the strategies for one of our scenarios. We have also shown how our strategies can be adapted for a larger search area consisting of adjoined discrete, symmetric regions. The same controller can then be used (modulo a symmetry transformation) on each region in turn until all objects have been located.

One limitation of this work is that we have only compared our approach with a simple search strategy. In the future we plan to compare our approach with alternative optimised search strategies. In addition, there are clearly scalability issues with our approach as the models generated can have hundreds of millions of states for simple scenarios. Therefore, to analyse real-world applications, abstraction (and refinement) techniques are required. In particular, we will investigate the use of the game-based abstraction approach of [KKNP10] in this context, as well as further symmetry reduction techniques [MDC06] as there is symmetry both in the environment, e.g. in a grid structure, and between agents. Regarding the formal models and specifications, improving the efficiency of the POMDP implementation in PRISM [NPZ17] could have significant modelling benefits, as in real applications control decisions must be based only on the information from sensors, and therefore only on a partial view of the environment. Stochastic games are also required to model and separate the nondeterminism present in the environment from the choices of the controller. Combining these aspects will require the analysis of partially observable stochastic games which are harder to solve than POMDPs [CD14]. PRISM has support for multi-objective queries [EKVY08] and this will allow the synthesis of more specific controllers, e.g. that optimise the mission time while limiting both power consumption and failure, and ensuring safety requirements.

As for using PRISM for the analysis, the current way optimal strategies are exported can be improved. In particular, having a graphical representation would have simplified the analysis. In addition, allowing the analysis of a synthesised strategy directly would have saved considerable effort. Currently, to do this, the strategy has to be exported to a file and then imported back into PRISM (together with the state space and reward structures).

Although in this paper we have limited our approach to UAVs, our aim is to apply it to other autonomous systems, such as underwater and ground vehicles.

Acknowledgements. The work described in this paper was supported by EPSRC grants EP/N508792/1, EP/N007565 and EC/P51133X/1. We would like to thank Dave Anderson, Euan McGookin and Sandor Veres for discussions on the autonomous systems that inspired this paper.

References

[Ack13] E. Ackerman. NASA lets curiosity rover loose on Mars in autonomous driving mode. IEEE Spectrum, 29 August 2013.

[AH99] R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999.

[ASK04] A. Agrawal, G. Simon, and G. Karsai. Semantic translation of simulink/stateflow models to hybrid automata using graph transformations. In *Proc. Int. Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT'04)*, volume 109 of *ENTCS*, pages 43–56, 2004.

[BBB+12] J. Barnat, J. Beran, J. Brim, T. Kratochvíla, and P. Ročkai. Tool chain to support automated verification of avionics Simulink designs. In *Proc. Int. Workshop on Formal Methods for Industrial Critical Systems (FMICS'12)*, volume 7436 of *LNCS*, pages 78–92, 2012.

[BBFL18] G. Bacci, P. Bouyer, U. Fahrenberg, and K. Larsen. Optimal and robust controller synthesis. In *Proc. Int. Symp. Formal Methods (FM'18)*, volume 10951 of *LNCS*, pages 2013–221. Springer, 2018.

[BBH+13] J. Barnat, L. Brim, V. Havel, J. Haveliček, J. Kriho, M. Lenčo, P. Ročkai, V. Štill, and J. Weiser. DiVinE 3.0 - an explicit-state model checker for multithreaded C & C++ programs. In *Proc. Int. Conf. Computer Aided Verifiication (CAV'13)*, volume 8044 of *LNCS*, pages 863–868, 2013.

[BCD+07] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and Didier D. Lime. UPPAAL-TIGA: Time for playing games! In *Proc. Int. Conf. Computer Aided Verification (CAV'07)*, volume 4590 of *LNCS*, pages 121–125. Springer, 2007.

[BDH+07] J. Burdick, N. DuToit, A. Howard, C. Looman, J. Ma, and T Wongpiromsarn. Sensing, navigation and reasoning technologies for the DARPA urban challenge. In *DARPA Urban Challenge Final Report, Tech. Rep.*, 2007.

[BMS04] S. Bouabdallah, P. Murrieri, and R. Siegwart. Design and control of an indoor micro quadrotor. In *Proc. Int. Conf. Robotics and Automation (ICRA'04)*, pages 4393–4398. IEEE, 2004.

[CCGR99] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A new symbolic model verifier. In *Proc. Int. Conf. Computer Aided Verification (CAV'99)*, pages 295–499, 1999.

[CD14] K. Chatterjee and L. Doyen. Partial-observation stochastic games: How to win when belief fails. *ACM Trans. Computational Logic*, 15(2):16:1–16:44, 2014.

[CK00] A. Chutinan and B. Krogh. Verification of infinite-state dynamic systems using approximate quotient transition systems. *IEEE Trans. Automatic Control*, 46(9):101–109, 2000.

[CKNZ11] E. Clarke, W. Klieber, M. Nováček, and P. Zuliani. Model checking and the state explosion problem. In *Tools for Practical Software Verification: LASER, International Summer School*, volume 7682 of *LNCS*, pages 1–30. Springer, 2011.

[DDL+12] A. David, D. Du, K. Larsen, A. Legay, A. Mikučionis, M. Poulsen, and S. Sedwards. Statistical model checking for stochastic hybrid systems. In *Proc. Int. Workshop on Hybrid Systems and Biology (HSB'12)*, volume 92 of *EPTCS*, pages 122–136, 2012.

[DFK+15] K. Draeger, V. Forejt, M. Kwiatkowska, D. Parker, and M. Ujma. Permissive controller synthesis for probabilistic systems. *Logical Methods in Computer Science*, 11(2), 2015.

[DFL+16] L. Dennis, M. Fisher, N. Lincoln, A. Lisitsa, and S. Veres. Practical verification of decision-making in agent-based autonomous systems. *Automated Software Engineering*, 23(3):1–55, 2016.

[DH04] J. Dabney and T. Harman. *Mastering Simulink*. Pearson/Prentice Hall, 2004.

[DMP09] A. Donaldson, A. Miller, and D. Parker. Language-level symmetry reduction for probabilistic model checking. In *Proc. Int. Conf. Quantitative Evaluation of Systems (QEST'09)*, pages 289–298. IEEE, 2009.

[DSBR14] X. Ding, S. Smith, C. Belta, and D. Rus. Optimal control of Markov decision processes with linear temporal logic constraints. *IEEE Trans. Automatic Control*, 59:1244–1257, 2014.

[DSL09] A. Das, K. Subbarao, and F. Lewis. Dynamic inversion with zero-dynamics stabilisation for quadrotor control. *Control Theory & Applications, IET*, 3(3):303–314, 2009.

[EKVY08] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. *Logical Methods in Computer Science*, 4:1–21, 2008.

[EP18] ECI-PwC. Flying high: Drones to drive jobs in the construction sector. *Presented at the National Conference of the Engineering Council of India (ECI)*, 2018.

[FDW13] M. Fisher, L. Dennis, and M. Webster. Verifying autonomous systems. *Communications of the ACM*, 56(9):84–93, 2013.

[FIS19] M. Foughali, F. Ingrand, and C. Seceleanu. Statistical model checking of complex robotic systems. In *Proc. Int. Symp. Model Checking of Software (SPIN'19)*, volume 11636 of *LNCS*, pages 114–134, 2019.

[FKNP11] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.

[FMM+16] P. Filipovikj, N. Mahmud, R. Marinescu, C. Seceleanu, O. Ljungkrantz, and Lönn H. Simulink to UPPAAL statistical model checker: Analyzing automotive industrial systems. In *Proc. Int. Symp. Formal Methods (FM'16)*, volume 9995 of *LNCS*, pages 748–756, 2016.

[FT15] J. Fu and U. Topcu. Computational methods for stochastic control with metric interval temporal logic specifications. In *Proc. Int. Conf. Decision and Control (CDC'15)*, pages 7440–7447. IEEE, 2015.

[FWHT15] L. Feng, C. Wiltsche, L. Humphrey, and U. Topcu. Controller synthesis for autonomous systems interacting with human operators. In *Proc. Int. Conf. Cyber-Physical Systems (ICCPS'15)*, pages 70–79. ACM, 2015.

[GHI+18] R. Giaquinta, R. Hoffmann, M. Ireland, A. Miller, and G. Norman. Strategy synthesis for autonomous agents using PRISM. In *Proc. NASA Formal Methods Symp. (NFM'2018)*, volume 10811 of *LNCS*, pages 220–236. Springer, 2018.

[GL19] T. Gibson-Robinson and G. Lowe. Symmetry reduction in CSP model checking. *International Journal on Software Tools for Technology Transfer*, 21:567–605, 2019.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proc. IEEE*, 79(9):1305–1320, 1991.

[Hen96] T. Henzinger. The theory of hybrid automata. In *Proc. Int. Symp. Logic in Computer Science (LICS'96)*, pages 278–292. IEEE, 1996.

[HIM+16] R. Hoffmann, M. Ireland, A. Miller, G. Norman, and S. Veres. Autonomous agent behaviour modelled in PRISM: A case study. In *Proc. Int. Symp. Model Checking Software (SPIN'16)*, volume 9641 of *LNCS*, pages 104–110. Springer, 2016.

[Hsu16] J. Hsu. U.S. navy's drone boat swarm practices harbor defense. IEEE Spectrum, 19 December 2016.

[Ire14] M. Ireland. *Investigations in Multi-Resolution Modelling of the Quadrotor Micro Air Vehicle*. PhD, University of Glasgow, 2014.

[IVA15] M. Ireland, A. Vargas, and D. Anderson. A comparison of closed-loop performance of multirotor configurations using non-linear dynamic inversion control. *Aerospace*, 2(2):325–352, 2015.

[JYL+16] Y. Jiang, Y. Yang, H. Liu, H. Kong, M. Gu, J. Sun, and L. Sha. From Stateflow simulation to verified implementation: a verified approach and a real-time train controller design. In *Proc. Int. Real-Time and Embedded Technology and Applications Symp. (RTAS'16)*, pages 1–11. IEEE, 2016.

[KEPS99] S Kowalewski, S Engell, J Preußig, and O Stursberg. Verification of logic controllers for continuous plants using timed condition/event-system models. *Automatica*, 35(3):505–518, 1999.

[KKNP10] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods in System Design*, 36:246–280, 2010.

[KMP10] Y. Kubera, P. Mathieu, and S. Picault. Everything can be agent! (extended abstract). In *Proc. Int. Conf. Autonomous Agents and Multi-Agent Systems (AAMAS10)*, pages 1547–1548, 2010.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. Int. Conf. Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[KP13] M. Kwiatkowska and D. Parker. Automated verification and strategy synthesis for probabilistic systems. In *Proc. Int. Symp. Automated Technology for Verification and Analysis (ATVA'13)*, volume 8172 of *LNCS*, pages 5–22. Springer, 2013.

[KP16] N. Kalra and S. Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193, 2016.

[KPR16] N. Kamaleson, D. Parker, and J. Rowe. Finite-horizon bisimulation minimisation for probabilistic systems. In *Proc. Int. Symp. Model Checking Software (SPIN'16)*, volume 9641 of *LNCS*, pages 147–164. Springer, 2016.

[KPW17] M. Kwiatkowska, D. Parker, and C. Wiltsche. PRISM-games: Verification and strategy synthesis for stochastic multi-player games with multiple objectives. *International Journal on Software Tools for Technology Transfer*, 20(2):195–210, 2017.

[KSK76] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. Springer, 1976.

[KWT11] H. Kress-Gazit, T Wongpiromsarn, and U. Topcu. Correct, reactive, high-level robot control. *IEEE Robotics and Automation Magazine*, 18(3):65–74, 2011.

[LAB15] M. Lahijanian, S. Andersson, and C. Belta. Formal verification and synthesis for discrete-time stochastic systems. *IEEE Trans. Automatic Control*, 60:2031–2045, 2015.

[LFD+18] M. Luckcuck, M. Farrell, L. Dennis, C. Dixon, and M. Fisher. Formal specification and verification of autonomous robotic systems: A survey. *CoRR*, abs/1807.00048, 2018.

[LK16] M. Lahijanian and M. Kwiatkowska. Specification revision for Markov decision processes with optimal trade-off. In *Proc. Int. Conf. Decision and Control (CDC'16)*. IEEE, 2016.

[LKM+08] F. Lerda, J. Kapinski, H. Maka, E. Clarke, and B. Krogh. Model checking in-the-loop. In *Proc. American Control Conference (ACC'08)*, pages 2734–2740, 2008.

[LPH17] B. Lacerda, D. Parker, and N. Hawes. Multi-objective policy generation for mobile robots under probabilistic time-bounded guarantees. In *Proc. Int. Conf. Automated Planning and Scheduling (ICAPS'17)*, pages 504–512. AAAI, 2017.

[LPY97] K. Larsen, P. Petterson, and W. Yi. UPPAAL in a nutshell. *Software Tools for Technology Transfer*, pages 134–152, 1997.

[MBR06] B. Meenakshi, A. Bhatnagar, and S. Roy. Tool for translating Simulink models into input language of a model checker. In *Proc. Int. Conf. Formal Engineering methods (ICFEM'06)*, LNCS, pages 606–620, 2006.

[MD14] M. Mueller and R. D'Andrea. Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers. In *Proc. Int. Conf. Robotics and Automation (ICRA'14)*, pages 45–52. IEEE, 2014.

[MDC06] A. Miller, A. Donaldson, and M. Calder. Symmetry in temporal logic model checking. *Computing Surveys*, 36, 2006.

[Mil09] S. Miller. Bridging the gap between model-based development and model checking. In *Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09)*, LNCS, pages 443–453, 2009.

[MMBC11] K. Manamcheri, S. Mitra, S. Bak, and M. Caccamo. A step towards verification and synthesis from Simulink/S-

tateflow models. In *Proc. Int. Conf. Hybrid systems: Computation and Control (HSCC'11)*, pages 317–318. ACM, 2011.

[NPZ17] G. Norman, D. Parker, and X. Zou. Verification and control of partially observable probabilistic systems. *Real-Time Systems*, 53:354–402, 2017.

[Rot16] J. Rothwell. US military tests 'Sea Hunter,' world's largest unmanned ship, amid 'deep concern' about China's naval expansion. The Telegraph, 3 May 2016.

[SCL+15] M Svoreňová, M Chmelík, K Leahy, H Eniser, K Chatterjee, I Černá, and C Belta. Temporal logic motion planning using POMDPs with parity objectives: Case study paper. In *Proc. Int. Conf. Hybrid Systems: Computation and Control (HSCC'15)*, pages 233–238. ACM, 2015.

[Sha53] L. Shapley. Stochastic games. In *Proc. National Academy of Science*, pages 1095–1100, 1953.

[Sha14] R. Sharan. *Formal methods for control synthesis in partially observed environments: application to autonomous robotic manipulation*. PhD thesis, California Institute of Technology, 2014.

[SKC+17] M. Svoreov, J. Ketnsk, M. Chmelk, K. Chatterjee, I. Cerna, and C. Belta. Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games. *Nonlinear Analysis: Hybrid Systems*, 23:230–253, 2017.

[SM17] S. Soudjani and R. Majumdar. Controller synthesis for reward collecting Markov processes in continuous space. In *Proc. Int. Conf. Hybrid Systems: Computation and Control (HSCC'17)*, pages 45–54. ACM, 2017.

[TM06] A. Tayebi and S. McGilvray. Attitude Stabilization of a VTOL Quadrotor Aircraft. *IEEE Transactions on Control Systems Technology*, 14(3):562–571, 2006.

[Tre17] HM Treasury. Autumn budget 2017: 25 things you need to know. UK Government website, 22 November 2017.

[VNE+01] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. The CLARAty architecture for robotic autonomy. In *Proc. Int. Conf. Aerospace (AeroConf'01)*, pages 121–132. IEEE, 2001.

[Voo09] H. Voos. Nonlinear control of a quadrotor micro-UAV using feedback-linearization. In *Proc. Int. Conf. Mechatronics*, pages 1–6. IEEE, 2009.

[Wil16] J. Wilson. Drones hacked and crashed by research team to expose design flaws. Engineering and Technology, 9 June 2016.

[WTM12] E. Wolff, U. Topcu, and R. Murray. Robust control of uncertain Markov decision processes with temporal logic specifications. In *Proc. Int. Conf. Decision and Control (CSC'12)*, pages 3372–3379. IEEE, 2012.

[YT16] D. Yadron and D. Tynan. Tesla driver dies in first fatal crash while using autopilot mode. The Guardian, 30 June 2016.