



## Research Article

# A particle swarm optimization approach using adaptive entropy-based fitness quantification of expert knowledge for high-level, real-time cognitive robotic control

Deon de Jager<sup>1</sup>  · Yahya Zweiri<sup>1</sup> · Dimitrios Makris<sup>1</sup>

Received: 29 July 2019 / Accepted: 14 November 2019 / Published online: 26 November 2019

© The Author(s) 2019 

## Abstract

High-level, real-time mission control of semi-autonomous robots, deployed in remote and dynamic environments, remains a challenge. Control models, learnt from a knowledgebase, quickly become obsolete when the environment or the knowledgebase changes. This research study introduces a cognitive reasoning process, to select the optimal action, using the most relevant knowledge from the knowledgebase, subject to observed evidence. The approach in this study introduces an adaptive entropy-based set-based particle swarm algorithm (AE-SPSO) and a novel, adaptive entropy-based fitness quantification (AEFQ) algorithm for evidence-based optimization of the knowledge. The performance of the AE-SPSO and AEFQ algorithms are experimentally evaluated with two unmanned aerial vehicle (UAV) benchmark missions: (1) relocating the UAV to a charging station and (2) collecting and delivering a package. Performance is measured by inspecting the success and completeness of the mission and the accuracy of autonomous flight control. The results show that the AE-SPSO/AEFQ approach successfully finds the optimal state-transition for each mission task and that autonomous flight control is successfully achieved.

**Keywords** High-level robot control · Cognitive robotics · Knowledge optimization · Maximum entropy principle · Markov decision process · Adaptive entropy-based fitness quantification · Set-based particle swarm optimization

## 1 Introduction

Cognitive robotics is described as “the study of knowledge representation and reasoning problems, faced by an autonomous robot (or agent) in a dynamic and uncertain world” [1]. The efficient control of robots, operating in remote environments, has long proved to be a challenge, especially when the robot operates in a dynamic environment. In a typical remote-controlled, semi-autonomous robot, elementary knowledge is stored in a knowledgebase (KB), which is augmented by a domain expert. The KB is used for the generation of inference models (controllers) for decision-making by the robot.

Arguably, the most widely used statistical formalism for the high-level control of robots is the use of finite state automata (FSA) as high-level controllers. In FSA, Markov decision processes (MDPs) represent the states and state-action pairs of each state-transition. The state-action pairs are referred to as the policies of the FSA and it is the policies which governs the behaviour of the robot. The objective is to find an optimum policy for each state-transition. However, the generation of inference models in dynamic and remote environments is not trivial.

✉ Deon de Jager, K0952100@kingston.ac.uk | <sup>1</sup>Faculty of Science, Engineering and Computing, Kingston University London, London SW15 3DW, UK.



### 1.1 Problem description

- Augmenting or modifying the KB of a remotely-deployed robot becomes more convoluted, error-prone and computationally expensive if the structure of the KB is complex. Limited communication bandwidth also restricts the maintenance of the KB.
- High-level controllers are often generated through machine learning techniques, where an FSA is generated for all states. These techniques progressively learn the policies of the FSA, using user-defined parameters, which are often selected subjectively or derived through experimentation. For example, Q-learning uses a user-defined probability state-transition matrix (STM) and reinforcement learning (RL) uses a user-defined discounted reward to statistically determine the best policies for the FSA. Changes in the environment, is likely to lead to the re-optimization of the parameters and re-learning of the model.
- When machine learning is used to generate models as high-level controllers, the controller (FSA) is learnt in its entirety. For dynamic environments, a large number of models have to be learnt to handle different scenarios. However, when the underlying knowledgebase changes or the environment changes, learnt models may become obsolete and need to be replaced. Due to the time it takes to relearn a model, re-generation of high-level controllers in real-time operation becomes infeasible.

### 1.2 Proposed solution

The solution proposed in this research study aims to reduce and simplify the “cognitive world” of the robot (UAV in this study). In this study, cognitive reasoning is defined as decision-making, based on current knowledge which is optimized using real-time evidence. The solution introduces a cognitive reasoning process (CRP) for the UAV in order to govern its behaviour in real-time. This process is similar to that of the human cognitive framework [2] and intuitively provides the UAV with some intelligence in decision-making. Figure 1, gives an overview of the CRP:

A domain expert provides all the knowledge, i.e. the KB, mission definitions and evidence definitions, to be used by the CRP. The KB is the set of all state-transitions (policies) between the states of the UAV. Each state-transition has a trigger formula which is composed of a set of conjunctive propositions. The mission definition is defined as a set of tasks to be completed in order to successfully complete the specified mission. The evidence definition is a set of variables, representing environmental observations, received in real-time. The KB, mission definitions and

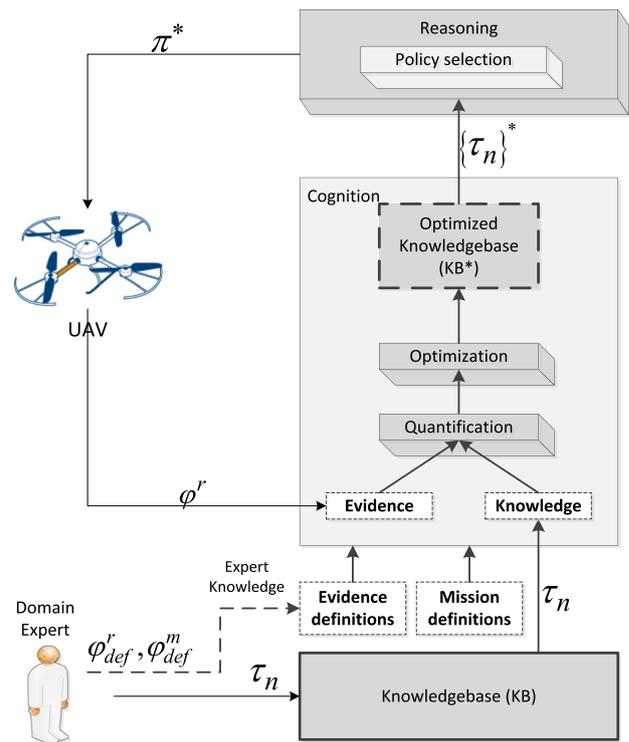


Fig. 1 Cognitive reasoning process (CRP)

evidence definitions are used by the CRP to find the optimal policy, given the evidence. The CRP will be discussed in detail in Sect. 4.

Since the KB is a set of discrete elements, the problem is defined as a constrained, set-based knowledge optimization problem. Therefore, the CRP uses the AE-SPSO and AEFQ algorithms to find, quantify and evaluate each potential policy for optimality. The best policy is then selected and its action is passed as a command to the UAV.

To the best of our knowledge, there has been no attempt to affect high-level robot control through dynamic and real-time policy optimization using a particle swarm optimization approach.

The main contributions of this research study are the following:

1. The introduction of a novel adaptive entropy fitness quantification (AEFQ) algorithm for the statistical quantification of state-transitions (policies) of a FSA as high-level controller.
2. The introduction of an improved set-based PSO which uses the AEFQ, to produce an optimized KB\*, which contains the optimal policies for selection.
3. The generation of re-usable high-level controllers (FSAs) as a result of the real-time, evidence-based, policy optimization and execution of the CRP.

The remainder of the research study is organized as follows: Sect. 2 reviews related work on high-level autonomous control in robotics; Sect. 3 provides some theoretical background on the methodologies used in the study. Section 4 introduces the methodology for the CRP and discusses the novel AEFQ algorithm and AE-SPSO algorithms in detail. Sections 5, 6 describes the experiment setup, results and analysis, respectively. Section 7 concludes the work of this research study and proposes relevant future work on autonomous high-level control through knowledge optimization in the CRP.

## 2 Related work

Improving the cognitive ability of robots has received a considerable amount of attention over the last decade. As described in the introduction, the two key factors of cognitive robotics, are knowledge representation and cognitive reasoning [3]. Although the problem of improving autonomy is non-trivial, it is relevant to a variety of robotic applications, for example, in humanoid robotics [4, 5], human–robot interaction [6–9], Search and Rescue (SAR) [10] and multi-robot systems [11].

### 2.1 Knowledge representation

In order to perform efficient inference, the structure and content of the KB is very important, especially if the KB has to be maintained remotely.

For many years various machine learning approaches, such as statistical relational learning (SRL) [12], inductive logic programming (ILP) [13, 14] and knowledge-based model construction (KBMC) [15, 16] have been used to derive expert knowledge from existing data sources. Some machine learning systems have been developed to learn and formulate knowledge: FOIL [17] learns Horn clauses from relational data and MADDEN [16] performs statistical knowledge extraction from textual data. CLAUDIEN [14] is an ILP engine which computes a set of logically valid clauses from data sets.

Horn clauses are particularly useful, as its form is similar to programmatic conditional statements, and therefore easier to implement.

More recent approaches are proving more suitable for cognitive robotics. Linear temporal logic (LTL) is used as a formal language to define the tasks of a robot, as applied in [18], where LTL is combined with Petri Nets to determine optimal movement planning for multiple robots.

The problem of high-dimensionality in the relationship between task planning, using LTL and robot motion is investigated by Shoukry et al. [19]. Here, LTL is used to define a set of propositions, applicable to all robots, for

each region of the workspace. The robots' movements across regions are controlled by the LTL propositions.

### 2.2 Cognitive reasoning

A lot of research have been focussed on the low-level control of robots, for example improving path planning in dynamic environments, where obstacles are avoided by prioritizing and predicting the future behaviour of the object [20]. However, cognitive robotics is concerned with cognitive reasoning, using current knowledge. In [9], a semi-autonomous high-level controller is proposed for the semi-autonomous control of robot teams in urban search and rescue missions. The objective of the controller is to reduce the workload of the robot operator. Other cognitive robotic approaches, for example, inductive logic programming (ILP) which is used for predicate generation, is combined with reinforcement learning (RL) to learn optimal behavioural policies in [21].

A popular approach used for cognitive robotics is the combination of (LTL) and MDPs, where the LTL formulae provide a formal definition of tasks for the robot and the MDPs are used to synthesize high-level controllers. However, synthesizing high-level controllers in a, dynamic environment remains a challenge. Meyer and Dimarogonas [22] introduces a framework to increase the adaptability of the synthesis process, by using a 3-layer top-down hierarchical decomposition of the control problem. A three step-approach is used to firstly, solve the LTL problem on a FSA, secondly, find the best policy for transitioning and thirdly, synthesize a controller.

Reinforcement learning (also referred to as Q-learning) of MDP type controllers are increasingly being combined with other methodologies to learn high-level controllers to accomplish some task. Generally, the objective of Q-learning is to iteratively select the best policy, i.e. state-action, which maximizes the expected discounted reward (Q-value), given the current state, the user-defined STM and user-defined rewards. The most popular approach is the use of the Bellman equations [23], which calculates the optimal Q-value over all policies. In [24], Q-learning is used in combination with a Deep Deterministic Policy Gradients (DDPG) algorithm for a UAV to learn a landing task in simulation. In [25], the effectiveness of the Q-learning algorithm for robot path planning, is improved by using a flower pollinating algorithm to initialize the q-values of the algorithm.

### 2.3 Critical review

For semi-autonomous robots, remotely deployed in unknown and dynamic environments, the complexity of the formulation of expert knowledge is critical. The

methods discussed in Sect. 2.1 may prove to be sufficient for discovery and formulation of knowledge for high-level robot control in a controlled or well-defined environment. However, in an unknown or highly dynamic environment, the knowledge needs to be maintained in real-time, often over vast distances. For these types of environments, the methods discussed will prove to be computationally expensive and error-prone and may be constrained by communication bandwidths.

Similarly, learning high-level controllers using machine learning techniques, such as Q-learning mentioned in Sect. 2.2, will be sufficient in well-defined or controlled environments. For environments such as these, effective models can be learned. A degree of dynamism may be catered for by learning a large number of models to cater for as many scenarios as possible. However, if the environment is unknown or highly dynamic, the accuracy of the models will be sub-optimal. Moreover, it is infeasible to relearn a model in real-time, every time the environment changes.

The AE-SPSO/AEFQ methodology introduced in this study follows a real-time optimization approach, while a machine learning methodology follows an a priori learning approach. The main differences between the two methodologies are:

- The modification and updating of the expert-knowledge (missions, states and rules), which is used in the optimization process is simplified, and therefore uses less bandwidth and is less error-prone.
- No subjective user-defined state-transition probabilities are required; instead probabilities are accurately calculated from real-time evidence, received from the environment.
- An open world assumption (OWA) (c.f. Sect. 3.1) is used in the quantification process.
- State-transitions can be composite, i.e. multiple state-transitions between the same two states, each statistically quantified using its own rules and with its own actions.
- A re-usable high-level controller (FSA) is dynamically created, step-by-step in real-time.

These differences are too significant to do a fair empirical comparison between a machine learning approach and the AE-SPSO/AEFQ approach proposed in this study. This is mainly due to the differences between the learning approach followed by a machine learning methodology and a real-time optimization approach followed the AE-SPSO/AEFQ approach.

Therefore, the performance of the AE-SPSO/AEFQ methodology will be evaluated experimentally, on two simulated UAV benchmark problems.

## 3 Background

In this section, some characteristics of the knowledgebase are discussed and an overview of the standard particle swarm optimization algorithm and the set-based particle swarm optimization algorithms is given.

### 3.1 Knowledgebase characteristics

The KB contains domain-specific and relevant knowledge, required for decision-making, i.e. inference. In this study, the KB is composed of the set of all state-transitions with a trigger formula for each state-transition. The trigger formula is composed of a conjunctive set of logic propositions. Inference, is defined as the CRP which uses the KB to find the optimal policy to control the UAV. The CRP is directly influenced by the completeness and consistency of the KB.

The KB is considered to be complete, if all possible knowledge needed for inference is formulated within the KB. This means each trigger formula is completely defined.

The KB is considered to be consistent, if there are no changes made to any of the ground propositions during inference. While consistency is not a requirement for the representation of the knowledge in the KB, it is important for efficient reasoning.

In addition, inference is performed, using either a closed world assumption (CWA) or an open world assumption (OWA) [26, 27]. The CWA assumes the knowledge about the environment is complete and consistent. This means that, unless it is known that a trigger formula is true, it must be assumed to be false.

The OWA on the other hand, assumes that the knowledge representing the environment is incomplete or inconsistent. This means that if all knowledge is not explicitly specified, the truth of the trigger formula is considered unknown, but not false. Therefore, a probability ("degree of belief") is statistically allocated to the trigger formula.

In a dynamic environment, the KB can never be complete, as the domain expert could update the KB with new or updated state-transitions or trigger formulae at any time.

Also, the KB can never be consistent, as the evidence, which will be applied to the trigger formula for fitness quantification, constantly changes in real time.

Clearly, for a robot functioning in a dynamic and uncertain environment, the OWA is the preferred approach to follow. Therefore, the solution must enable simple, dynamic and efficient maintenance of the KB and dynamic and real-time reasoning for high-level robot control. Section 4 below discusses the proposed methodology to achieve this, in more detail.

### 3.2 Overview of standard particle swarm optimization

Particle swarm optimization (PSO) is a stochastic optimization algorithm, which has been successfully applied to optimization problems in the fields of engineering and robotics [28–30]. PSO has been successfully applied to problems where the search space is either continuous or discrete.

Inspired by the movement and behaviour of a flock of birds searching for food, Eberhart and Kennedy introduced the standard particle swarm optimization (StdPSO) algorithm [31]. The swarm of particles moves through a D-dimensional solution space. The position of particle  $i$  in the solution space represents a candidate solution, which is defined as a solution vector,  $\mathbf{X}_i \in \mathbb{R}^D$ . The optimality of the candidate solution is determined by a fitness function,  $f(\mathbf{X}_i) \in \mathbb{R}$ . The particle's velocity represents the step size and direction of its movement and is defined by a vector  $\mathbf{v}_i \in \mathbb{R}^D$ . StdPSO iteratively updates each particle's velocity and position using the following equations:

$$v_{ij}(t + 1) = \omega v_{ij}(t) + c_1 r_{1j} (y_{ij}(t) - x_{ij}(t)) + c_2 r_{2j} (\hat{y}_j(t) - x_{ij}(t)) \tag{1}$$

$$x_{ij}(t + 1) = x_{ij}(t) + v_{ij}(t + 1) \tag{2}$$

where  $v_{ij}(t)$  represents the  $j$ th element of the velocity vector of particle  $i$ , at the  $t$ th iteration. An inertia weight  $\omega$  is applied to the particle velocity. Two key components of the velocity equation are, the cognitive component,  $c_1 r_{1j} (y_{ij}(t) - x_{ij}(t))$ , and the social component,  $c_2 r_{2j} (\hat{y}_j(t) - x_{ij}(t))$ , where  $y_{ij}(t)$  represents the  $j$ th element of the personal best vector of particle  $i$  at the  $t$ th iteration and  $\hat{y}_j(t)$  represents the  $j$ th element of the global best vector of the swarm at the  $t$ th iteration. The term,  $x_{ij}(t)$ , represents the  $j$ th element of the current position of particle  $i$  at the  $t$ th iteration. The two positive real numbers  $c_1$  and  $c_2$  are acceleration constants, used to scale the contributions of the cognitive and social components. The random values,  $r_{1j}, r_{2j} \sim U(0, 1)$ , add a stochastic element to the cognitive and social components. A user-defined inertia weight,  $\omega$ , is added to the current velocity [32], which, along with the acceleration constants, balances the effect between global search and local search.

The general fitness function for the PSO is defined as:

$$f : \mathbb{R}^{n\phi} \rightarrow \mathbb{R} \tag{3}$$

For a minimization problem, the personal best position at the next iteration is calculated as,

$$y_i(t + 1) = \begin{cases} y_i(t) & \text{if } f(X_i(t + 1)) \geq f(y_i(t)) \\ X_i(t + 1) & \text{if } f(X_i(t + 1)) < f(y_i(t)) \end{cases} \tag{4}$$

and for a maximization problem, the personal best position at the next iteration is calculated as,

$$y_i(t + 1) = \begin{cases} y_i(t) & \text{if } f(X_i(t + 1)) \leq f(y_i(t)) \\ X_i(t + 1) & \text{if } f(X_i(t + 1)) > f(y_i(t)) \end{cases} \tag{5}$$

For a minimization problem, the global best position at the next iteration is calculated as,

$$\hat{y}(t + 1) = \begin{cases} \hat{y}(t) & \text{if } f(X_i(t + 1)) \geq f(\hat{y}(t)) \\ X_i(t + 1) & \text{if } f(X_i(t + 1)) < f(\hat{y}(t)) \end{cases} \tag{6}$$

and for a maximization problem, the global best position at the next iteration is calculated as,

$$\hat{y}(t + 1) = \begin{cases} \hat{y}(t) & \text{if } f(X_i(t + 1)) \leq f(\hat{y}(t)) \\ X_i(t + 1) & \text{if } f(X_i(t + 1)) > f(\hat{y}(t)) \end{cases} \tag{7}$$

On conclusion of all iterations, all (or most) of the particles have converged on the best solution, which is represented by the global best vector.

### 3.3 Overview of set-based particle swarm optimization

When the search space is discrete, the velocity and position update Eqs. (1) and (2) cannot be used without redefinition. Langeveld and Engelbrecht [33] introduced a generic, set-based PSO (SPSO), suitable for optimization problems with a discrete search space. To remain in accordance with standard PSO, velocity and position update equations, are redefined in terms of set-based operators. The set-based velocity equation is:

$$V_i(t + 1) = (c_1 r_1 \otimes (Y_i(t) \ominus X_i(t))) \oplus (c_2 r_2 \otimes (\hat{Y}(t) \ominus X_i(t))) \oplus (c_3 r_3 \odot_k^+ A_i(t)) \oplus (c_4 r_4 \odot^- S_i(t)) \tag{8}$$

where  $(c_1 r_1 \otimes (Y_i(t) \ominus X_i(t)))$  is the cognitive component and  $(c_2 r_2 \otimes (\hat{Y}(t) \ominus X_i(t)))$  is the social component. Two additional components are added to the standard PSO equation:  $c_3 r_3 \odot_k^+ A_i(t)$  and  $c_4 r_4 \odot^- S_i(t)$ , where  $A_i(t) = U \setminus (X_i(t) \cup Y_i(t) \cup \hat{Y}(t))$  and  $S_i(t) = (X_i(t) \cap Y_i(t) \cap \hat{Y}(t))$ . The parameters are  $c_1, c_2 \in [0, 1]$  and  $c_3, c_4 \in [0, |U|]$ . The random numbers,  $r_1$  to  $r_4$ , are random values sampled from a uniform distribution, i.e.  $r_1, r_2, r_3, r_4 \sim \Omega(0, 1)$ .

The set-based position update equation is:

$$X_i(t + 1) = X_i(t) \boxplus V_i(t + 1) \tag{9}$$

While the movement of particles through the search space is governed by Eqs. (1) and (2) for a continuous search space, Eqs. (8) and (9) govern the movement of particles (sets) through a discrete set-based search space.

The knowledge optimization process of the CRP will be implemented using an improved SPSO algorithm.

### 4 Methodology

This CRP of an autonomous UAV (illustrated in Fig. 1), have two main functions: cognition and reasoning. The cognition component is tasked with the statistical optimization of the knowledge from the KB, given the environmental evidence. The reasoning component is tasked with selecting the optimal action  $\pi^*$ , from the optimized knowledgebase, KB. The methodology is implemented in Algorithms 1–3.

#### 4.1 Evidence definitions

Environmental evidence is defined as two sets, representing the mission parameters and evidence parameters. The former is provided by the domain expert in relation to a specific mission and the latter is sensory information observed by the robot.

The runtime parameter is defined as,

$$\Phi^r = \{ \varphi_1^r, \varphi_2^r, \dots, \varphi_{n_{\Phi^r}}^r \} \tag{10}$$

where  $\varphi_i^r, i = 1, \dots, n_{\Phi^r}$  is the runtime parameter representing the evidence observed in the environment.

The mission parameters are defined as,

$$\Phi^m = \{ \varphi_1^m, \varphi_2^m, \dots, \varphi_{n_{\Phi^m}}^m \} \tag{11}$$

where  $\varphi_j^m \in [lb^m, ub^m], j = 1, \dots, n_{\Phi^m}$  defines the mission parameter, constrained to specified lower and upper boundaries. Both the mission and evidence parameters are used in the calculation of constraint averages, and represent the dynamic environmental information received by the robot's CRP.

#### 4.2 Knowledgebase definitions

The KB is defined as the set of state-transitions which governs the behaviour of the UAV:

$$KB = \{ \tau_1, \tau_2, \dots, \tau_{n_{|KB|}} \} \tag{12}$$

where  $\tau_k \in KB, k = (1, \dots, |KB|)$  represents a state-transition in the KB.

The state-transition is a tuple,

$$\tau_k = (\nu, \eta, \mathcal{A}, \mathcal{F})$$

where  $\nu = \{0, 1\}$  indicates whether the transition is valid,  $\eta \in \mathbb{Z}^+$  is an objective identifier assigned to the

transition,  $\mathcal{A} = \{a_1, \dots, a_{n_A}\}$  is a set of actions and  $\mathcal{F} = \{p_1, p_2, \dots, p_{n_F}\}$  is the trigger formula for the transition, consisting of a set of simple logic propositions.

Each proposition  $p_l \in \mathcal{F}, l = (1, \dots, n_F)$  is defined by a domain expert and is a tuple,

$$p_l = (\varphi_i^r, logical\_operator, \varphi_j^m) \tag{13}$$

where the runtime and mission parameters are related by a logical\_operator, from the set  $\{>, <, =\}$ , to form simple propositions of the form:

$$(\varphi_i^r > \varphi_j^m), (\varphi_i^r < \varphi_j^m) \text{ and } (\varphi_i^r = \varphi_j^m) \tag{14}$$

(Any non-numeric argument is discretized to a numeric value, prior to quantification of  $\mathcal{F}$ ).

The indicator  $\nu$ , the objective identifier  $\eta$ , the actions  $\mathcal{A}$  and all the propositions  $p_l$  are defined and maintained by the domain expert.

#### 4.3 Adaptive entropy fitness quantification

##### 4.3.1 Model construction

In order to perform the quantification of a state-transition  $\tau_k$ , a problem-specific model is constructed before it is presented to the MEP equation for quantification. Given a state-transition  $\tau_k \in KB$  the model is formally defined as a tuple,

$$\mathcal{M}_{\tau_k} = (\mathbf{V}, \mathbf{X}, \mathbf{F}, \mathbf{\Lambda}) \tag{15}$$

The set of variables are represented by  $\mathbf{V} = \{ \{v^Q\} \cup \{v_1^P, v_2^P, \dots, v_{n_P}^P\} \cup \{v_1^A, v_2^A, \dots, v_{n_A}^A\} \}$  where  $v^Q$  is the query variable,  $v_p^P, p = 1, \dots, n_P$  is a predictor variable, representing a proposition in the trigger formula and  $v_l^A, l = 1, \dots, n_A$  is an association variable. Note that, since the propositions are independent, they will not have any effect on the query variable, unless relevant associations are defined between the query variable and appropriate predictor variables. The associations are problem-specific and are defined by the user.

Let  $m_{\tau_k} = \left| \{v^Q\} \cup \{v_1^P, v_2^P, \dots, v_{n_P}^P\} \right|$  and  $n_{\tau_k} = 2^{m_{\tau_k}}$ , then a  $m_{\tau_k} \times n_{\tau_k}$  constraint matrix.  $\mathbf{X}$  is the state space of the trigger formula and defines all the joint statements of  $\{v^Q\} \cup \{v_1^P, v_2^P, \dots, v_{n_P}^P\}$ . A binary constraint function,  $F(X = x_{ij}), i \in n_{\tau_k}$  and  $j \in m_{\tau_k}$  assigns a boolean constraint to each variable in the state space. Let  $n_V = (1 + n_P + n_A)$ , then vector  $\mathbf{F} = (F_1, F_2, \dots, F_{n_F}), n_F = n_V$  are constraint averages for each of the variables in  $\mathbf{V}$ . The vector  $\mathbf{\Lambda} = (\lambda_1, \lambda_2, \dots, \lambda_{n_A}), n_A = n_V$ , represents the Lagrange multipliers, calculated for each variable in  $\mathbf{V}$ , using (25).

Each constraint average  $F_{n_f} \in \mathbb{F}$  represents the degree of belief in a proposition and is derived from real-time information (evidence) received from the environment. The constraint average follows the OWA, and is crucial for the accurate quantification of the state-transition.

In this research study, the constraint average is calculated by interpreting a proposition as a degree of believe, (probability), derived from a distance calculation. For example, Fig. 2 illustrate two example state-transitions:

A constraint average for the proposition is calculated by measuring the progress of the current runtime parameter  $\varphi_i^r$ , relative to the operational bounds of the mission task. The result is a probability assigned to the proposition. Figure 3 illustrates the approach:

This approach ensures that the constraint average accurately reflects relevant environmental evidence. This also ensures that the fitness quantification of the trigger formula for the state-transition is based on relevant and correct environmental evidence.

The rule is translated into a probability as follows:

Firstly, given the proposition  $p_i$ , calculate the total operation distance  $d_j^m$ , using the upper and lower bounds of the mission argument:

$$d_j^m = ub_j^m - lb_j^m \tag{16}$$

Calculate the current distance  $d_i^r$  of the runtime argument,  $\varphi_j^r$  with respect to the upper and lower bounds of the mission parameter,  $\varphi_j^m$ , according to the logical operation of the proposition:

$$d_i^r = \begin{cases} \varphi_i^r - lb_j^m; & \text{if } p_i = (\varphi_i^r \leq \varphi_j^m) \\ ub_j^m - \varphi_i^r; & \text{if } p_i = (\varphi_i^r \geq \varphi_j^m) \\ 0; & \text{if } p_i = (\varphi_i^r \neq \varphi_j^m) \\ 1; & \text{if } p_i = (\varphi_i^r = \varphi_j^m) \end{cases} \tag{17}$$

Use (16) and (17) to calculate a real valued distance, in the range [0, 1], for the proposition:

$$Pr(p_i) = \frac{d_i^r}{d_j^m} \tag{18}$$

where  $Pr(p_i)$  represent the relative remaining distance of  $\varphi_i^r$ , within the boundaries  $lb_j^m$  and  $ub_j^m$  as a probability. Once the distances for each proposition have been calculated, the distances for each of the joint statements can be calculated. To illustrate, let  $v^Q = p_0$ ,  $v_1^P = p_1$  and  $v_2^P = p_2$ , then the state space consists of  $2^3 = 8$  joint statements.

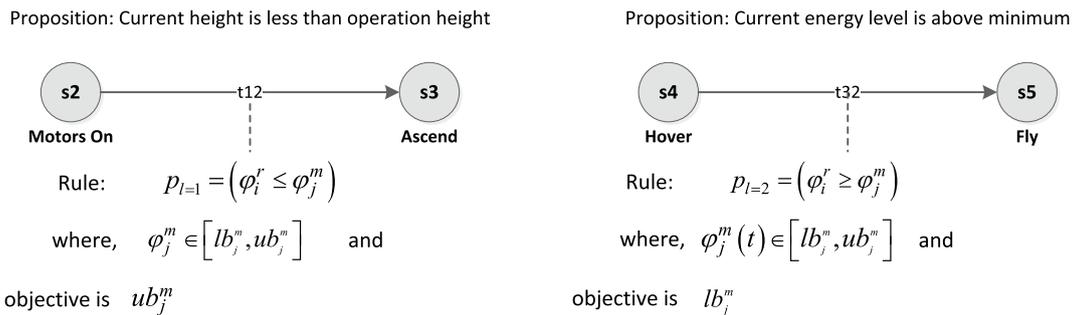


Fig. 2 Example state-transitions with corresponding propositions

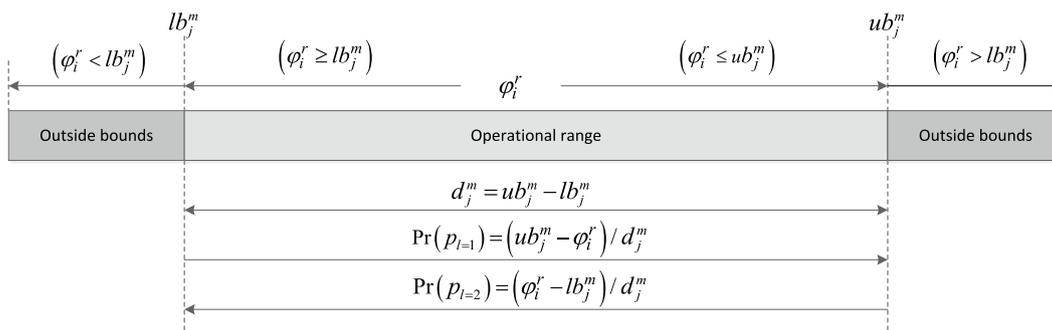


Fig. 3 Method for constraint average assignment to propositions

The joint distances, for the predictor variables are calculated as follows:

$$d_{p_1 p_2} = d_{p_1} + d_{p_2} \tag{19}$$

$$d_{p_1 \bar{p}_2} = d_{p_1} + (1 - d_{p_2}) \tag{20}$$

$$d_{\bar{p}_2 p_2} = (1 - d_{p_1}) + d_{p_2} \tag{21}$$

$$d_{\bar{p}_1 \bar{p}_2} = (1 - d_{p_1}) + (1 - d_{p_2}) \tag{22}$$

The overall distance  $d_f$ , represented by the probability distribution over all the propositions of the trigger formula, is calculated by:

$$d_f = (d_{p_1 p_2} + d_{p_1 \bar{p}_2} + d_{\bar{p}_2 p_2} + d_{\bar{p}_1 \bar{p}_2}) \tag{23}$$

With all the joint distances of the joint statements available, the respective constraint averages can now be calculated. Firstly, the constraint average  $F_1$  of the query variable  $p_0$  is set to 1.0. The constraint averages for the predictor and association variables are then set as follows:

$$\mathbf{F} = \left( d_{p_0}, \frac{(d_{p_1 p_2} + d_{p_1 \bar{p}_2})}{d_f}, \frac{(d_{p_1 p_2} + d_{\bar{p}_1 p_2})}{d_f}, \frac{(d_{p_1 p_2} + d_{p_1 \bar{p}_2})}{d_f}, \frac{(d_{p_1 p_2} + d_{\bar{p}_1 p_2})}{d_f}, \frac{d_{p_1 p_2}}{d_f} \right) \tag{24}$$

Next, the Lagrange multipliers are determined.

The duality between the Lagrange multipliers and the user-defined constraint averages, allows the Legendre transform to be used to derive the Lagrange multipliers:

$$\mathcal{L}_{trans} = \Lambda = \min_{\lambda_k} \left( \ln Z(\lambda_1, \lambda_2, \dots, \lambda_k) - \sum_{j=1}^{m_{\tau_k}} \lambda_j F_j \right) \tag{25}$$

The multipliers are derived by varying the values of  $\lambda_k$  while keeping the constraint average,  $F_j$  fixed, until  $\mathcal{L}_{trans}$  reaches a minimum. Table 1 shows an example of a model for a trigger formula containing two propositions, B and C, including associations with the query variable A, i.e. AB, AC, ABC. The model contains a  $m_{\tau_k} \times n_{\tau_k}$  boolean constraint matrix, where  $m_{\tau_k} = 3$  and  $n_{\tau_k} = 8$ .

Once the model is complete, the fitness quantification can be performed.

### 4.3.2 Fitness quantification

Given the model  $\mathcal{M}_{\tau_k}$ , the probability distribution,  $\mathbf{Q} = (q_1, q_2, \dots, q_{n_Q})$ ,  $n_Q = n_{\tau_k}$  over the variables (propositions) of the trigger formula can now be calculated. Given the  $m_{\tau_k} \times n_{\tau_k}$  constraint matrix and let  $i \in n_{\tau_k}$  and  $j \in m_{\tau_k}$ , the MEP is then formally defined as:

**Table 1** Model of illustrative example

$i$	A	B	C	AB	AC	ABC
1	1	1	1	1	1	1
2	1	1	0	1	0	0
3	1	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	1	0	0	0
6	0	1	0	0	0	0
7	0	0	1	0	0	0
8	0	0	0	0	0	0
	$\langle F_A \rangle$	$\langle F_B \rangle$	$\langle F_C \rangle$	$\langle F_{AB} \rangle$	$\langle F_{AC} \rangle$	$\langle F_{ABC} \rangle$
	$\lambda_1$	$\lambda_2$	$\lambda_3$	$\lambda_4$	$\lambda_5$	$\lambda_6$

$$(q_i | \mathcal{M}_{\tau_k}) = \frac{1}{Z(\lambda_1, \lambda_2, \dots, \lambda_k)} e^{-\sum_{j=1}^{m_{\tau_k}} \lambda_j F_j(X=x_i)} \tag{26}$$

$$\text{where } Z(\lambda_1, \lambda_2, \dots, \lambda_k) = \sum_{i=1}^{\tau_k} e^{\sum_{j=1}^{m_{\tau_k}} \lambda_j F_j(X=x_i)}$$

$Z$  is the partition function which ensures the probabilities are assigned between 0 and 1. The Lagrange multipliers

are represented by  $\lambda_j$ ,  $j = 1, \dots, k$  and  $F_j(X = x_i)$  assigns a real-world, domain-specific constraint, to the state  $i$  of variable  $j$ .

(Refer to [34], chapters 24 and 25 for a detailed discussion on the mathematical derivation of the Legendre transformation and the MEP formula).

Finally, the fitness of the state-transition  $\tau_k \in KB$  is calculated as,

$$f(\tau_k) = v \times q_1 \tag{27}$$

where  $v \in \tau_k$  and  $v = 1$  indicate a valid state-transition and  $v = 0$  indicate an invalid state-transition.

Note that any of the resulting probabilities (including marginal probabilities) in the distribution  $\mathbf{Q}$  may now be used in the fitness quantification. However, in this study, only  $q_1$  will be used for fitness quantification, since its value is conditioned on all the predictor variables, i.e. propositions.

### 4.4 The AE-SPSO algorithm

The AE-SPSO algorithm is an improved variant of the SPSO (c.f. Sect. 3.3). The AE-SPSO algorithm eliminates the random removal of (potentially good) solutions from the personal best- and global best sets. Moreover, an elitism

approach is added to the AE-SPSO to ensure the best performing elements are retained in a scalable manner. The AE-SPSO algorithm uses the same cognitive, social and positioning components as those in Eqs. (1) and (2). However, since the particle is a set of state-transitions, each with a trigger formula, the algebraic operations of Eqs. (1) and (2) are not suitable and have to be redefined as set-based operations:

Let:

$X_i$  be the current position of particle  $i$ , (i.e. the set of state-transitions,  $\tau_k$ ).

$Y_i$  be the personal best position of particle  $i$ , (i.e. the personal best set of state-transitions).

$\hat{Y}$  be the global best position of the swarm (i.e. the global best set of state-transitions).

$c_1, c_2$  be the cognitive and social accelerators respectively.

then

$$V_i(t + 1) = r_1 c_{cog}(d_{cog}) \cup r_2 c_{soc}(d_{soc}) \tag{28}$$

$$X_i(t + 1) = \max_{\epsilon}(X_i \cup v_i(t + 1)) \tag{29}$$

where

$$\text{Cognitive difference} : d_{cog} : Y_i \cup (X_i \setminus Y_i)$$

The difference between the particle’s personal best set  $y_i$  and the particle’s current set  $X_i$  is defined as the unification of  $Y_i$  and the set-theoretic difference between  $Y_i$  and  $X_i$ . That is, all the elements in the particle’s personal best set are retained and the elements in  $X_i$  which are not in  $Y_i$  are included in the difference set.

$$\text{Social difference} : d_{soc} : \hat{Y} \cup (X_i \setminus \hat{Y})$$

The difference between the swarm’s global best set  $\hat{Y}$  and particle’s current set  $X_i$  is defined as the unification of  $\hat{Y}$  and the set-theoretic difference between  $\hat{Y}$  and  $X_i$ . That is, all the elements the swarm’s best set is retained and the elements in  $X_i$  which are not in  $\hat{Y}$  are included in the difference set.

$$\text{Cognitive velocity} : v_{cog} : f(KB, r_1, c_1) \cup (d_{cog})$$

The cognitive velocity is derived by the union of  $c_1$  random elements selected from the KB and the cognitive difference set. For each  $c_1$ , a random integer value  $r_1$  is selected from the range  $[1, |KB|]$  and the element (state-transition) at index  $r_1$  is added to  $d_{cog}$ .

$$\text{Social velocity} : v_{soc} : f(KB, r_2, c_2) \cup (d_{soc})$$

The social velocity is derived by the union of  $c_2$  random elements from the KB and the social difference set. For each  $c_2$ , a random integer value  $r_2$  is selected from the range  $[1, |KB|]$  and the element (state-transition) at index  $r_2$  is added to  $d_{soc}$ .

$$\text{Particle velocity} : V_i(t + 1) = v_{cog} \cup v_{soc}$$

The resulting velocity  $V_i(t + 1)$  is the union of the elements of cognitive velocity  $v_{cog}$  and the elements of the social velocity  $v_{soc}$ .

$$\text{Particle position} : X_i(t + 1) = \max_{\epsilon}(X_i \cup V_i(t + 1))$$

Using the AEFQ algorithm (Algorithm 1), the fitness  $f(\tau_k)$ , where  $\tau_k \in X_i$ , is calculated. In order to preserve the fittest elements from one iteration to the next, an elitism parameter  $\epsilon$ , is introduced [48]. The elitist parameter  $\epsilon$  specifies the number of fittest elements to include in the particle’s new position set. The new position  $X_i(t + 1)$  is derived by selecting the top  $\epsilon$  (fittest) elements from the union of the current position  $x_i$  and the velocity  $V_i(t + 1)$ . The selection of the top  $\epsilon$  elements is denoted by  $\max_{\epsilon}(\cdot)$ .

Note the absence of the inertia weight applied to the particle’s current velocity. In the standard PSO, the inertia weight  $w$ , along with the accelerator constants  $c_1, c_2$  control the granularity of the exploration. In set-based PSO, the accelerator constants  $c_1, c_2$  control the granularity by specifying the size of the random set of new elements to be added. Similarly, the inertia weight  $w$ , would specify the size of the subset of elements (the inertia set) to be selected from the velocity set. However, it would serve no purpose to add the inertia set again, because when calculating the new position set, the velocity set is already added in full to the current position set. Therefore, when calculating the difference sets  $d_{cog}$  and  $d_{soc}$  at the next iteration, the new position already includes the velocity elements.

---

Algorithm 1. AEFQ algorithm

---

1:	Input : Runtime parameter $\Phi^r$	eq. (10)
2:	: Task parameter $\Phi^m$	eq. (11)
3:	: State-transition $\tau_k$	eq. (12)
4:		
5:	Output : Fitness quantification, $\Pi$	eq. (26)
6:		
7:	Begin	
8:	Initialize model $\mathcal{M}_{\tau_k}$ , given $\mathcal{F} \in \tau_k$	eq. (15)
9:	Calculate weighted constraint averages $\mathbf{F}$	eq. (24)
10:	Calculate Lagrange multipliers $\mathbf{\Lambda}$ , given $\mathbf{F}$	eq. (25)
11:	Calculate probability distribution $\mathbf{Q}$ , given $\mathbf{\Lambda}$	eq. (26)
12:	Calculate the fitness $\Pi$ , given $\mathbf{Q}$	eq. (27)
13:	Return $\Pi$	
14:	End	

---

The CRP is implemented according to the logical processes defined by Algorithms 1–3. Algorithm 1 shows the implementation of the adaptive entropy fitness quantification method.

Note that, for simplicity, the sensory input is processed as single set, rather than each individual input element. Prior to the constraint average calculation (line 9), the

arguments of the trigger formula of the state-transition are ground using the corresponding sensory input parameters. This automation of the grounding process simplifies modification or creation of new propositions.

Algorithm 2 shows the process for finding the optimal solution (state-transition), based on the fitness of the particle, determined by the AEFQ (Algorithm 1).

---

Algorithm 2. AE\_SPSO algorithm

---

1:	Input : Knowledgebase, KB	
2:	: Task parameters, $\Phi^m$	
3:	: Runtime parameters $\Phi^r$	
4:		
5:	Output : Optimal solution/s, $\Pi^*$	
6:		
7:	Begin	
8:	Initialize a swarm, $S$ of $N$ particles, each particle containing	
9:	$n$ randomly selected state-transitions, $\tau_k \in KB$	
10:		
11:	Repeat //Activate the swarm	
12:	For each particle $i$ in swarm $S$	
13:	For each state-transition $\tau_k \in i$	
14:	// Quantify the state-transition fitness	
15:	$f(\tau_k) = AEFQ(\tau_k, \Phi^m, \Phi^r)$	(Alg. 1)
16:	Next state-transition	
17:		
18:	Update particle's velocity $V_i$	eq. (28)
19:	Update particle's position $X_i$	eq. (29)
20:		
21:	Next particle	
22:	Until stopping condition	
23:		
24:	// Select the optimal state-transition/s	
25:	Set $\Pi^* = X_i(t + 1)$	
26:	End	

---

The CRP uses the optimal set of solutions (state-transitions) found by the AE-SPSO to select and execute the relevant actions.

The code base of the AirSim/Unity simulator is C++ and the CRP were implemented using C#/NET (providing simpler memory management and easier data structure manipulation). To simplify future deployment of the CRP

---

Algorithm 3. CRP algorithm

---

```

1: Input      : Knowledgebase, KB // Domain expert knowledge
2:           : Mission parameters,  $\Phi^m$  with tasks,  $\varphi_j^m$ 
3:           : Runtime parameters  $\Phi^r$  with evidence,  $\varphi_i^r$  //Defaults
4:
5: Begin
6:   // Execute the mission
7:   For each task,  $\varphi_j^m \in \Phi^m$ 
8:     Repeat
9:       Input all  $\varphi_i^r \in \Phi^r$  from sensory input
10:
11:        // Find the optimal solution/s
12:         $\Pi^* = AE\_SPSO(KB, \Phi^m, \Phi^r)$  (Alg. 2)
13:
14:        // Select and execute action/s from optimal solution/s
15:        For each action  $\tau_k \in \Pi^*$ 
16:          For each action  $\mathcal{A} \in \tau_k$ 
17:            For each action  $a_n \in \mathcal{A}$ 
18:              Execute action  $a_n$ 
19:            Next action
20:          Next state-transition
21:        Until Task completed
22:      Next Task
23: End
    
```

---

### 5 Experiment setup

The methodology is experimentally evaluated by simulation, where a UAV autonomously execute two benchmark missions, one simple and one more complex. The performance measures for each of the benchmark missions are:

1. Success—measured by inspecting the completeness of the learned FSA (digraph), for each mission and;
2. Reasoning—measured by inspecting the level of velocity control of the UAV, based on reasoning about the statistical fitness of each state-transition.

Hardware: The experiments were executed on an Intel i7 laptop computer with 2.97 GHz quad core CPU, 16 Gb RAM and an Intel HD Graphics 4000 video adapter.

Software: The experiments were performed using the AirSim/Unity simulation environment, running on the Microsoft Windows 8.1 operating system.

Figure 4 illustrates the simulation’s software architecture:

on a UAV platform, the CRP was functionally abstracted from the simulation environment. Integration between the AirSim/Unity simulator and the CRP was performed

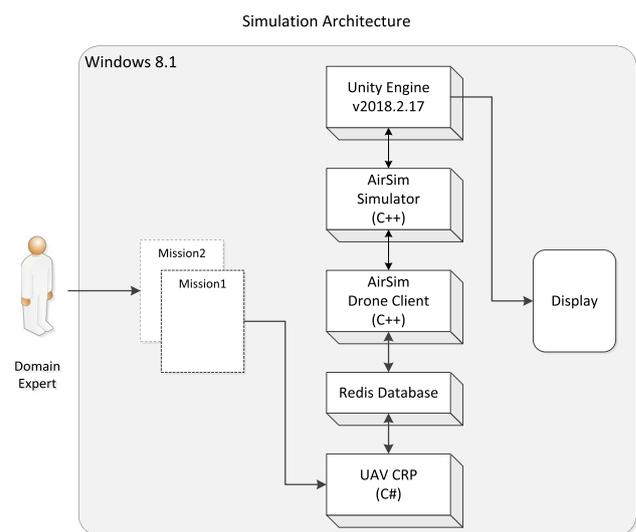


Fig. 4 Experiment platform architecture

**Fig. 5** Benchmark mission 1



**Fig. 6** Benchmark mission 2



**Table 2** UAV states

S1—Motors off	S2—Motors on	S3—Ascending
S4—Hovering	S5—Flying	S6—Descending
S7—Rotating	S8—Acquiring cargo	S9—Releasing cargo

using a Redis Cache database. Expert data, such as mission data, were entered in extensible markup language (XML) format.

### 5.1 Benchmark mission 1

From the Home (I) location, arm the motors, ascend to a specified operational height and fly to the Charging point (IV). Descend on the charging point and disarm the motors (Fig. 5).

### 5.2 Benchmark mission 2

From the Home (I) location, arm the motors, ascend to a specified operational height and fly to the Collection point (II). Descend and collect the cargo, then ascend to the specified operational height and fly to the Delivery point (III). Descend at the delivery point and deliver the cargo. Ascend to a new operational height and fly to the Charging point (IV) for recharging. Descend on the charging point and disarm the motors (Fig. 6).

The UAV platform has nine states (Table 2).

The nine UAV states yields a KB of 81 possible state-transitions (Table 3).

The KB is constructed as a square matrix, assuming a transition from every state to every other state. Valid states are defined by the domain expert, by setting an indicator on the state-transition as well as defining a trigger formula for each valid state-transition. Although some

**Table 3** UAV state-transitions

	s1	s2	s3	s4	s5	s6	s7	s8	s9
s1	t <sub>1</sub>	t <sub>2</sub>	t <sub>3</sub>	t <sub>4</sub>	t	t <sub>6</sub>	t <sub>7</sub>	t <sub>8</sub>	t <sub>9</sub>
s2	t <sub>10</sub>	t <sub>11</sub>	t <sub>12</sub>	t	t <sub>14</sub>	t <sub>15</sub>	t	t <sub>17</sub>	t <sub>18</sub>
s3	t <sub>19</sub>	t <sub>20</sub>	t <sub>21</sub>	t <sub>22</sub>	t <sub>23</sub>	t <sub>24</sub>	t <sub>25</sub>	t <sub>26</sub>	t <sub>27</sub>
s4	t <sub>28</sub>	t <sub>29</sub>	t <sub>30</sub>	t <sub>31</sub>	t <sub>32</sub>	t <sub>33</sub>	t <sub>34</sub>	t <sub>35</sub>	t <sub>36</sub>
s5	t <sub>37</sub>	t <sub>38</sub>	t <sub>39</sub>	t <sub>40</sub>	t <sub>41</sub>	t <sub>42</sub>	t <sub>43</sub>	t <sub>44</sub>	t <sub>45</sub>
s6	t <sub>46</sub>	t <sub>47</sub>	t <sub>48</sub>	t <sub>49</sub>	t <sub>50</sub>	t <sub>51</sub>	t <sub>52</sub>	t <sub>53</sub>	t <sub>54</sub>
s7	t <sub>55</sub>	t <sub>56</sub>	t <sub>57</sub>	t <sub>58</sub>	t <sub>59</sub>	t <sub>60</sub>	t <sub>61</sub>	t <sub>62</sub>	t <sub>63</sub>
s8	t <sub>64</sub>	t <sub>65</sub>	t <sub>66</sub>	t <sub>67</sub>	t <sub>68</sub>	t <sub>69</sub>	t <sub>70</sub>	t <sub>71</sub>	t <sub>72</sub>
s9	t <sub>73</sub>	t <sub>74</sub>	t <sub>75</sub>	t <sub>76</sub>	t <sub>77</sub>	t <sub>78</sub>	t <sub>79</sub>	t <sub>80</sub>	t <sub>81</sub>

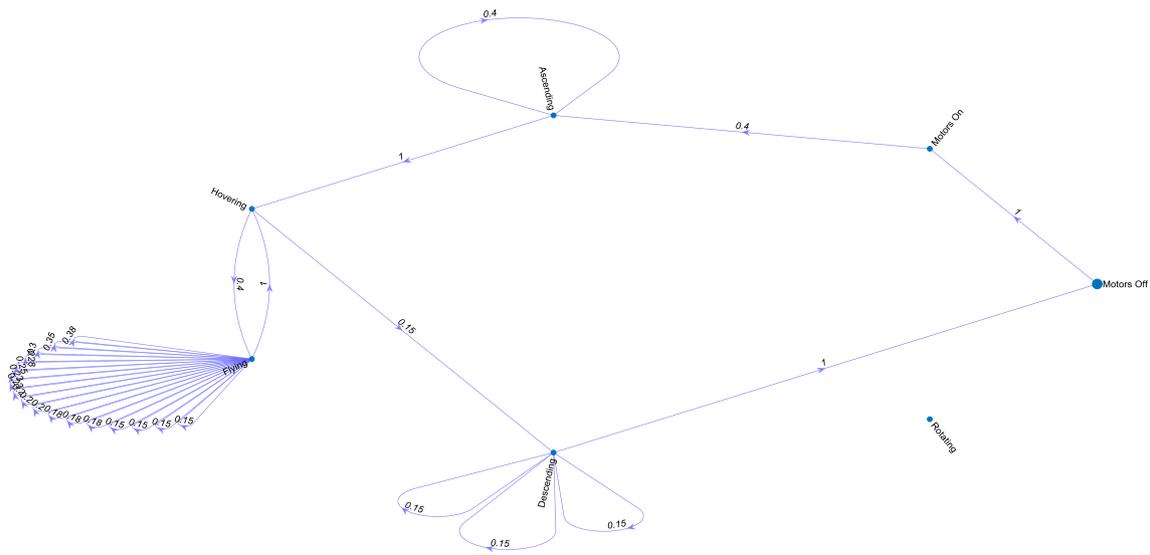


Fig. 7 FSA learned for benchmark mission 1

state-transitions can never be valid [e.g. from S5 (flying) to S1 (Motors Off)], this approach makes it simpler for the domain expert to activate new transitions. Valid state-transitions for the experiments are marked in Table 3 as shaded/italic/bold cells.

For each of the benchmark missions, an annotated video of the simulation is recorded and published to YouTube:

1. UAV benchmark mission 1 [35]
2. UAV benchmark mission 2 [36]

## 6 Experiment results

### 6.1 Benchmark mission 1: results

Figure 7 shows the resulting FSA for benchmark mission 1, dynamically learned by the CRP from the KB (Table 3), during the execution of mission 1. The node in bold shows the start state, i.e. Motors Off.

Figure 8 provides a “zoomed” view showing the “fly” state-transitions, and corresponding fitness of each, generated by the CRP.

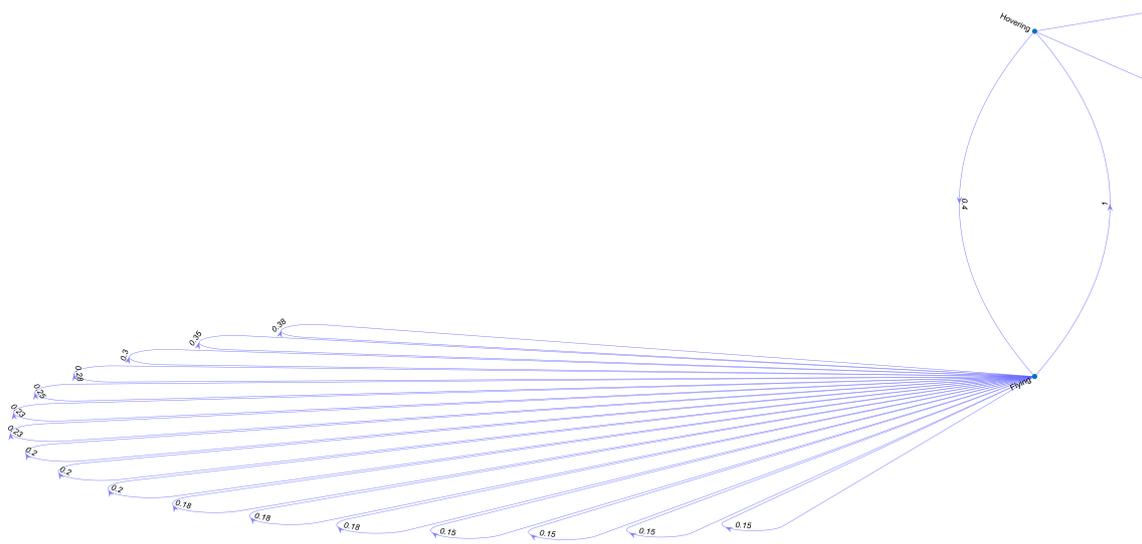


Fig. 8 A “zoomed” image of the iterative “fly” transition generated during mission 1

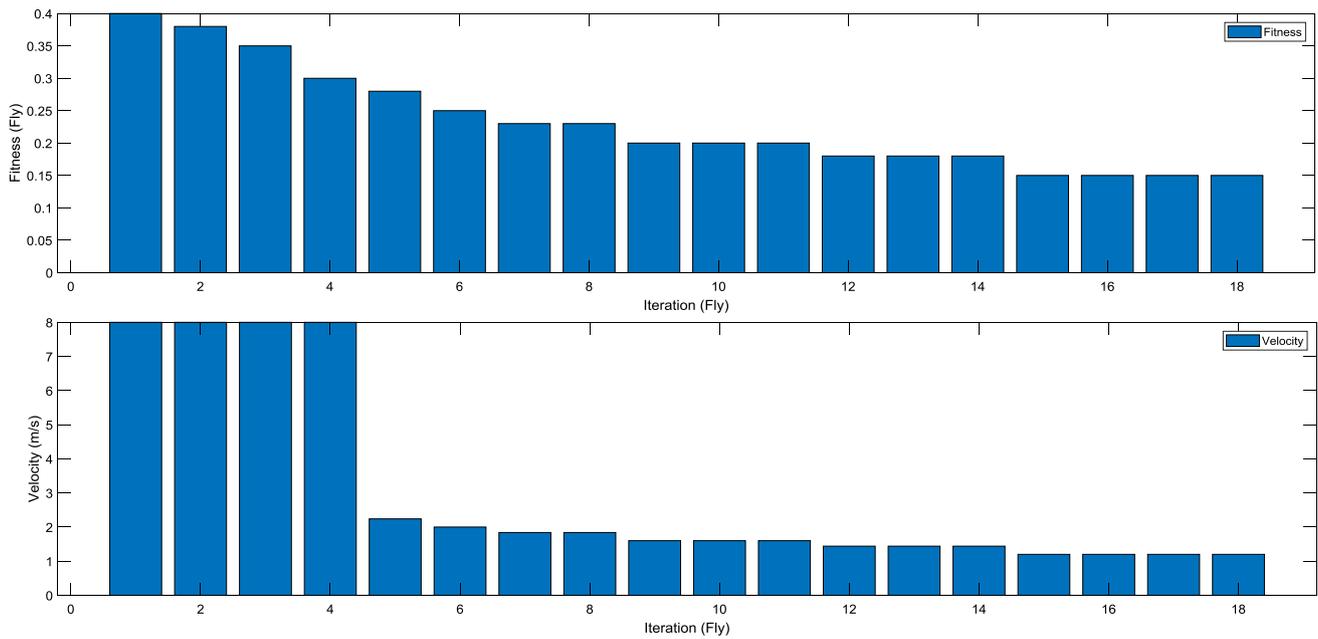


Fig. 9 Dynamic velocity adjustment during mission 1

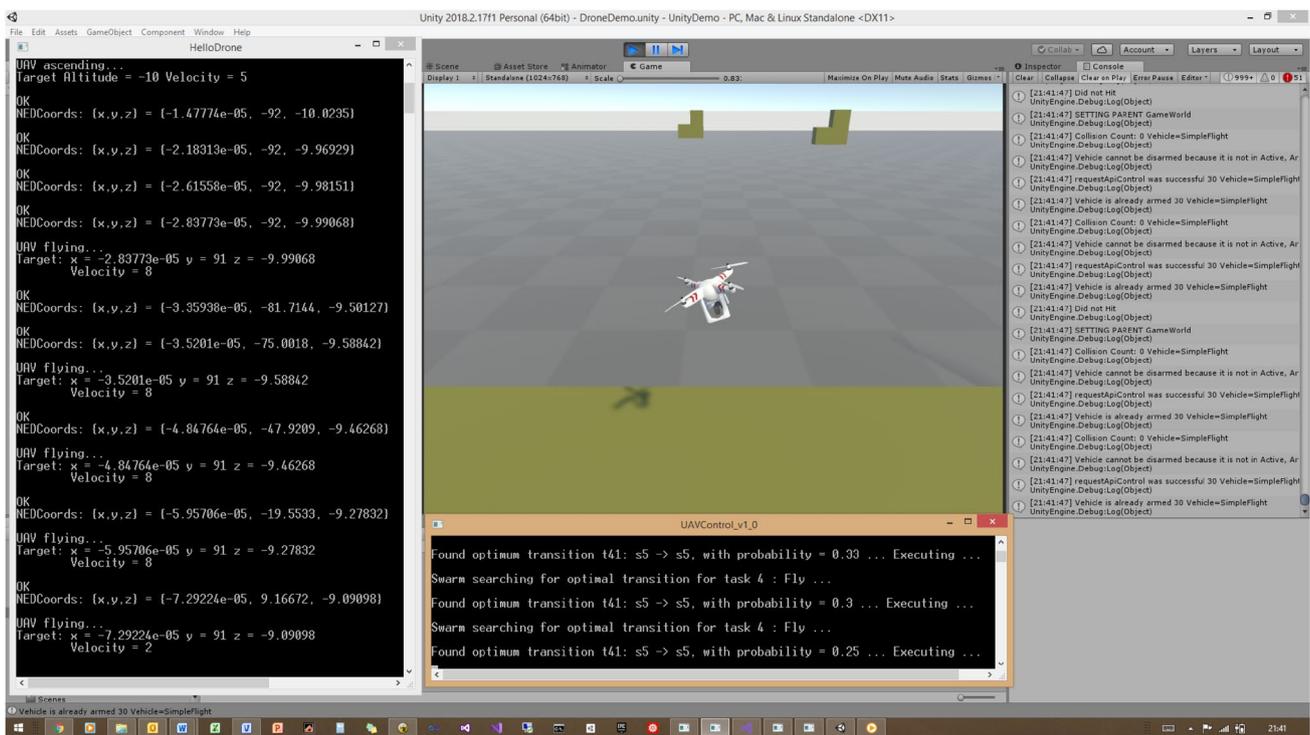


Fig. 10 UAV reducing its velocity as it approaches its target destination

The learned FSA can be saved and, provided the mission and operational conditions remain the same, may be used as high-level controller to execute similar, subsequent missions.

Figure 9 shows the dynamic control of the velocity, derived from the state-transition fitness. The graph shows the reduction in velocity, in accordance with the reduction in fitness of the “fly” state-transition, as the UAV nears its destination.

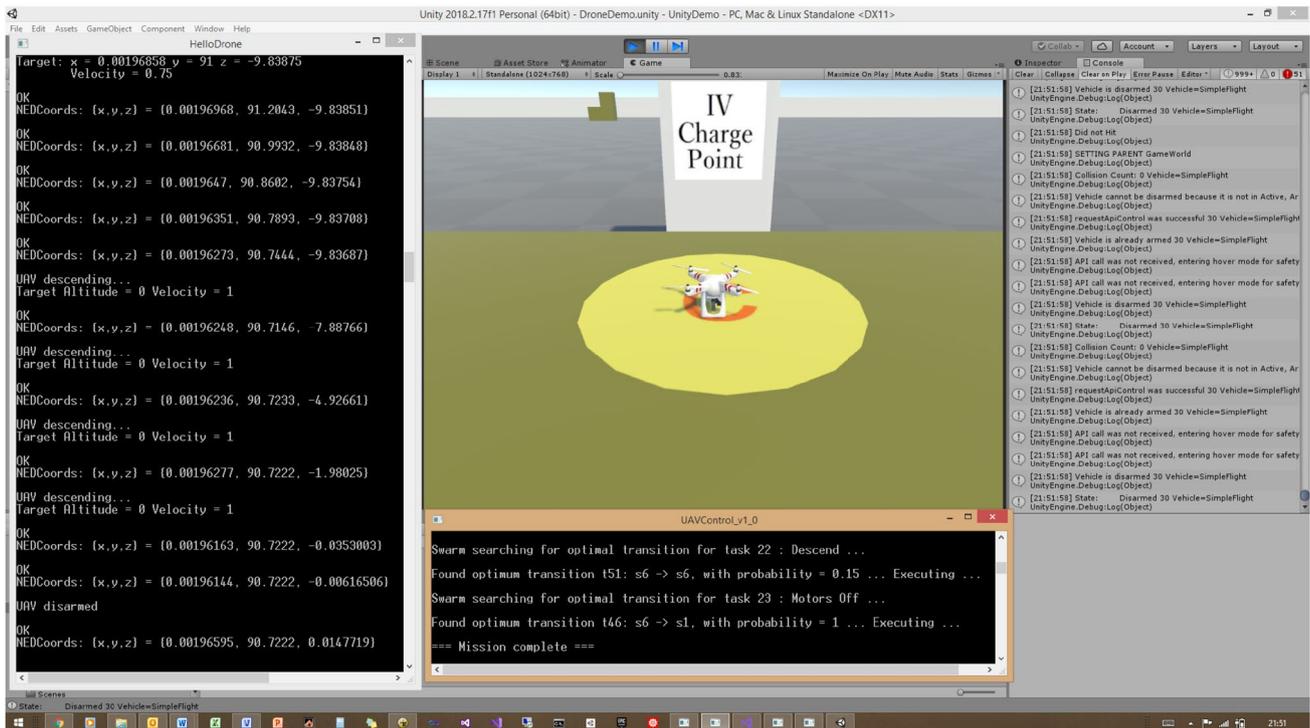


Fig. 11 UAV reaching its destination and completing the mission

On the graph, the target destination (charging) of mission 1 can be seen at task 18. The graph shows that the UAV proportionally reduces its velocity as it approaches its destination.

Figures 10 and 11 show some key stages in the simulation for benchmark mission 1. The window at the bottom shows the CRP finding the optimal state-transitions and sending the corresponding actions to the simulator. The window on the left shows the results of the simulator as it performs the actions received from the CRP.

Figure 10, shows the dynamic velocity adjustment of the UAV, derived from the fitness probability, as the UAV approaches its target. This behaviour is used to evaluate performance measure 2.

As the UAV approaches its target,  $\Pr(\varphi_i^r < \varphi_j^m)$  is reduced from 0.3 to 0.25 and the velocity of the UAV (indicated in the window left) is adjusted accordingly from 8.00 to 2.00 m/s.

Figure 11, shows the successful completion of the mission. This behaviour is used to evaluate performance measure 1.

When the UAV reached its target destination (the charging point), it descends and successfully completes the mission.

## 6.2 Benchmark mission 2: results

Figure 12 shows the resulting FSA for benchmark mission 2, dynamically learned by the CRP from the KB (Table 3), during the execution of the mission. The node in bold shows the start state, i.e. Motors Off.

Figure 13 provides a “zoomed” view showing the “fly” state-transitions, and corresponding fitness for each, generated by the CRP.

The graph in Fig. 14 shows the dynamic velocity control, derived from the state-transition fitness. The graph shows the corresponding reduction in velocity every time the UAV near its target.

On the graph, the three target destinations (collection, delivery and charging) of the missions can be seen at tasks 7, 15, and 21. The graph shows that the UAV proportionally reduces its velocity as it approaches each of the destinations.

Figures 15 and 16 shows some key stages in the simulation for benchmark mission 2. Figure 15 shows the UAV in process of collecting its cargo.

Figure 16 shows the UAV adjusting its velocity in accordance with the fitness of the state-transition, fly.

Figure 17 shows the UAV successfully delivering its cargo.

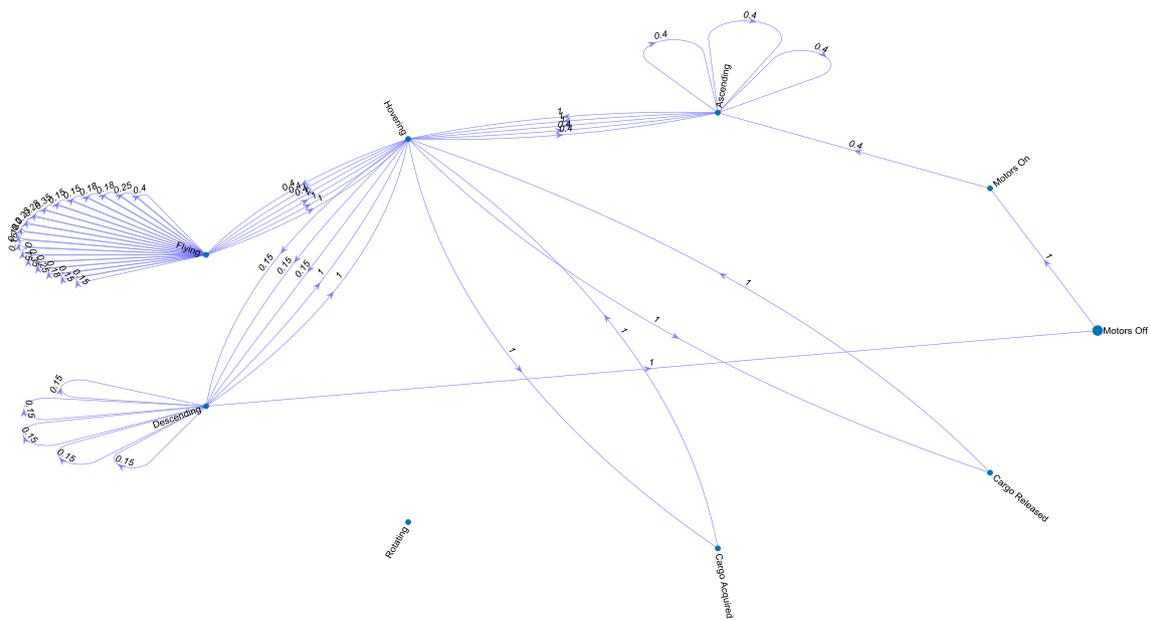


Fig. 12 FSA learned for benchmark mission 2

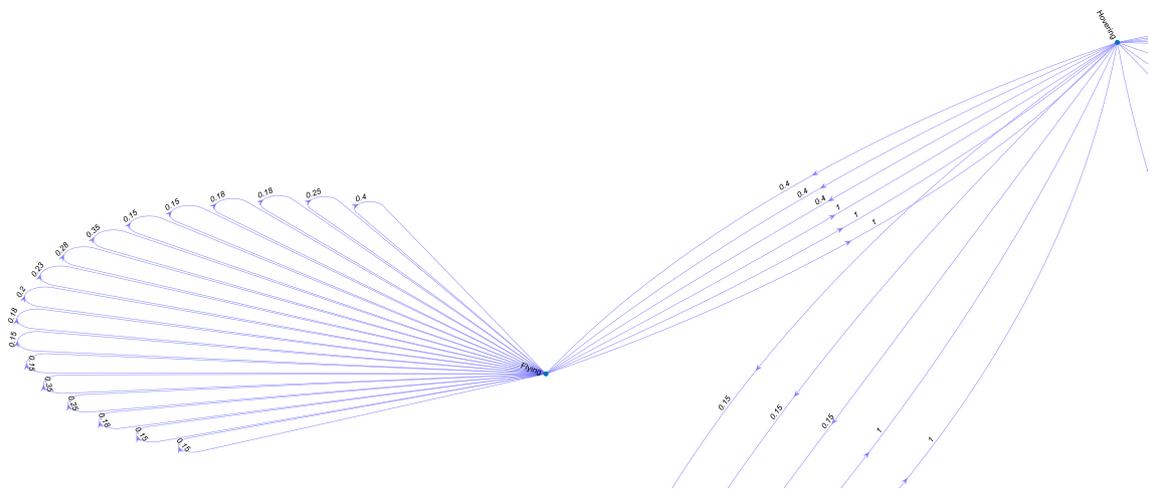


Fig. 13 A “zoomed” image of the iterative “fly” transition generated during mission 2

### 6.3 Discussion

#### 6.3.1 Performance

Overall, the experimental results (Figs. 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17) shows that our approach works well for real-time knowledge optimization for high-level control in cognitive robotics.

The simulation was executed repeatedly, with consistent results. Figures 7 and 8 (for benchmark mission 1) and Figs. 12 and 13 (for benchmark mission 2) shows that the approach successfully executed the expert-defined

missions. This success was also observed during the simulation. Figures 9 and 14, for benchmark mission 1 and 2 respectively, shows the successful reasoning for velocity control, using statistical reasoning. The figures show the corresponding velocity adjustment, based on the fitness (probability) which is also shown in the zoomed Figs. 8 and 13 for the “fly” action.

In addition, conducting the experiments also showed the following general benefits:

- The approach is less error prone and requires less bandwidth to maintain because, in our approach, knowl-

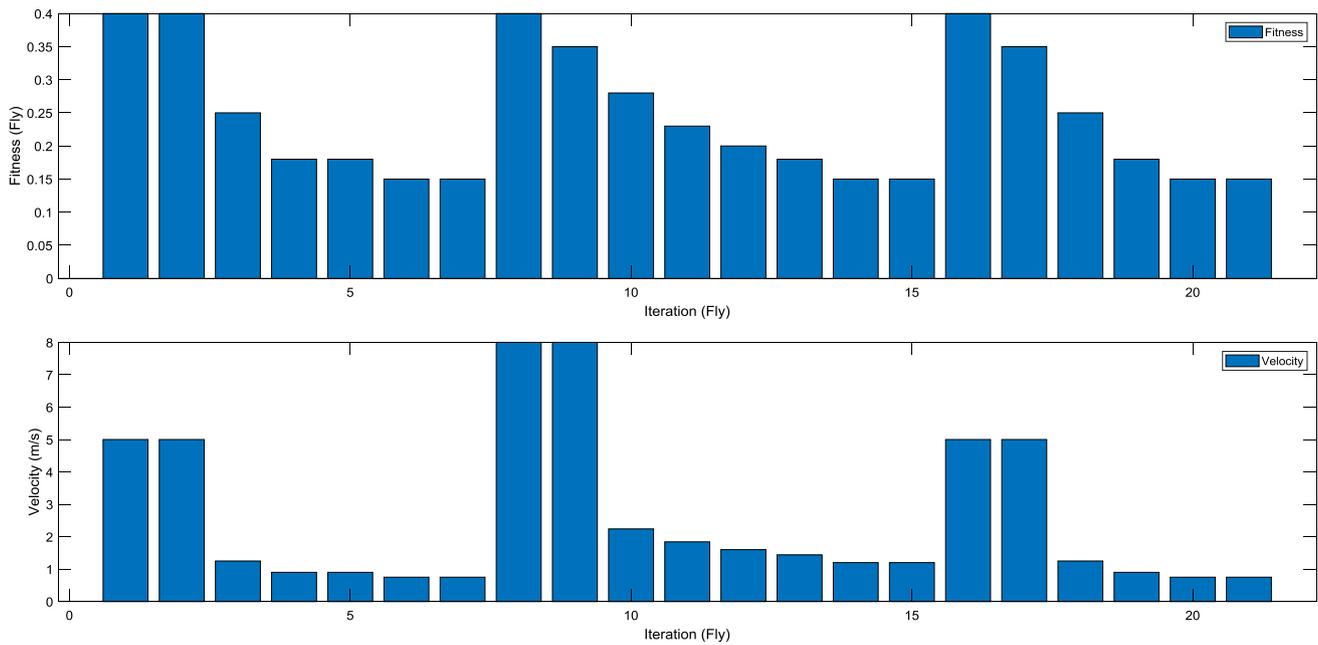


Fig. 14 Dynamic velocity adjustment during mission 2

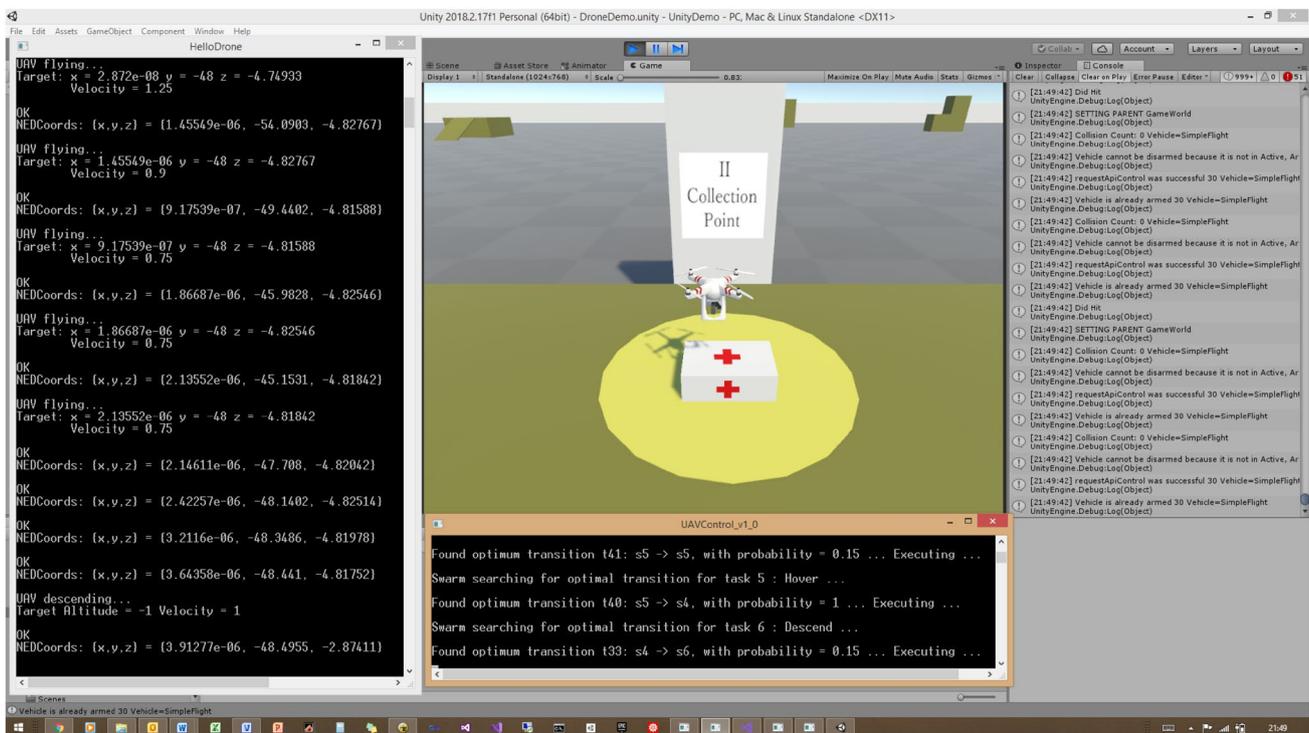


Fig. 15 UAV collecting its cargo at the collection point

edge and missions are defined using a simple structure. The trigger formula of state-transitions is constructed as a simple conjunction of propositions, and is therefore more intuitive to the domain expert. Moreover,

the knowledgebase and missions can be modified independently, reducing errors during the updating process.

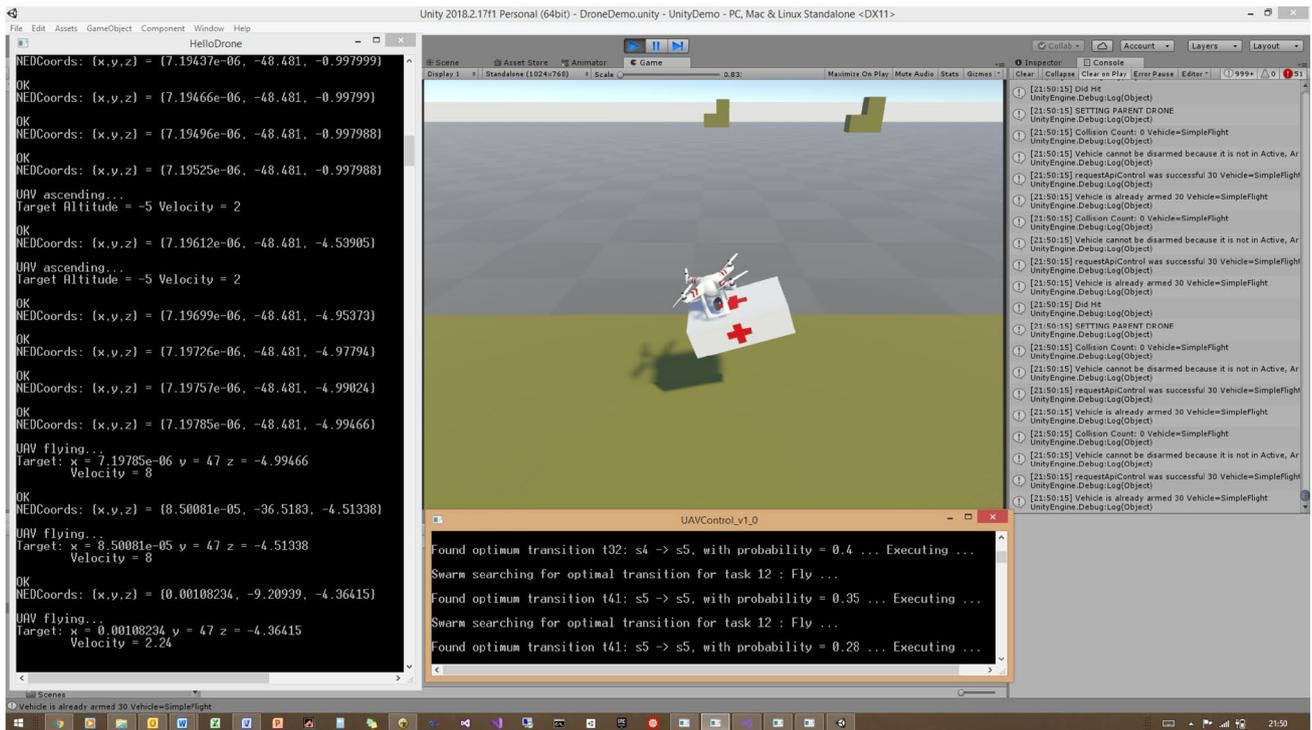


Fig. 16 UAV reducing its velocity as it approaches its target destination

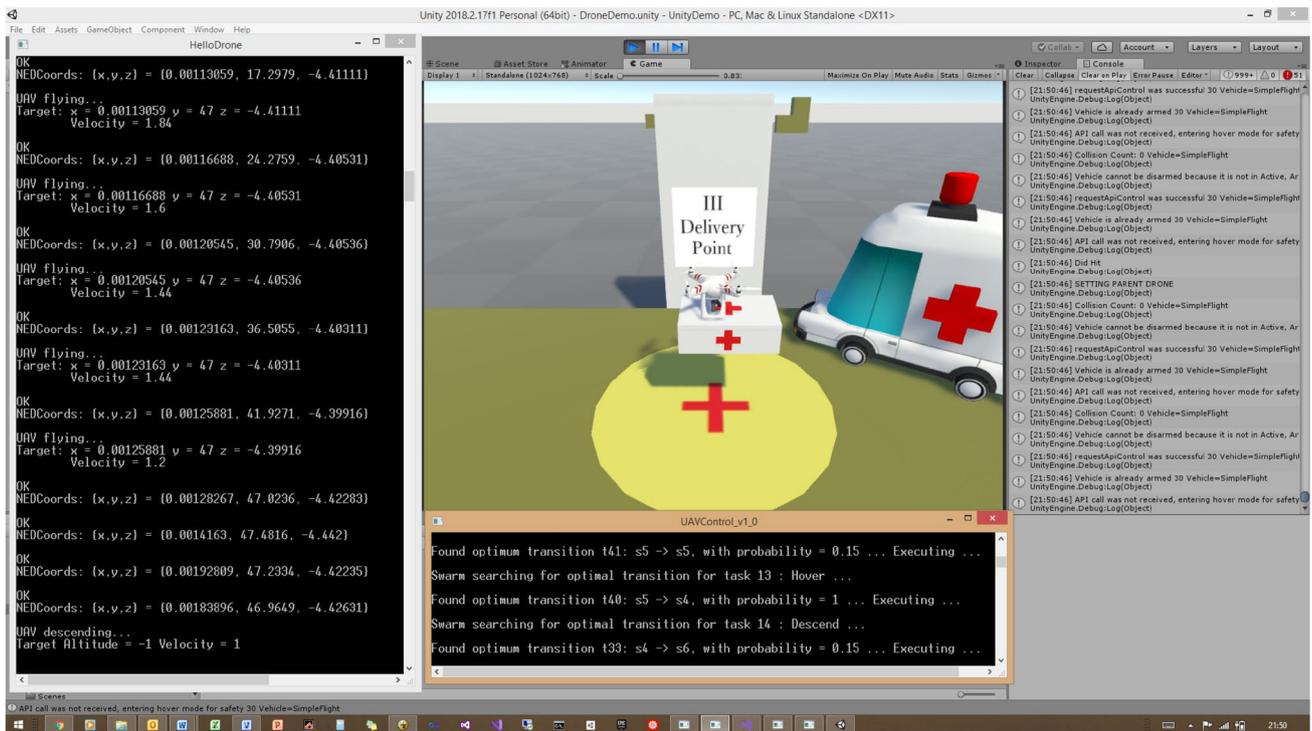
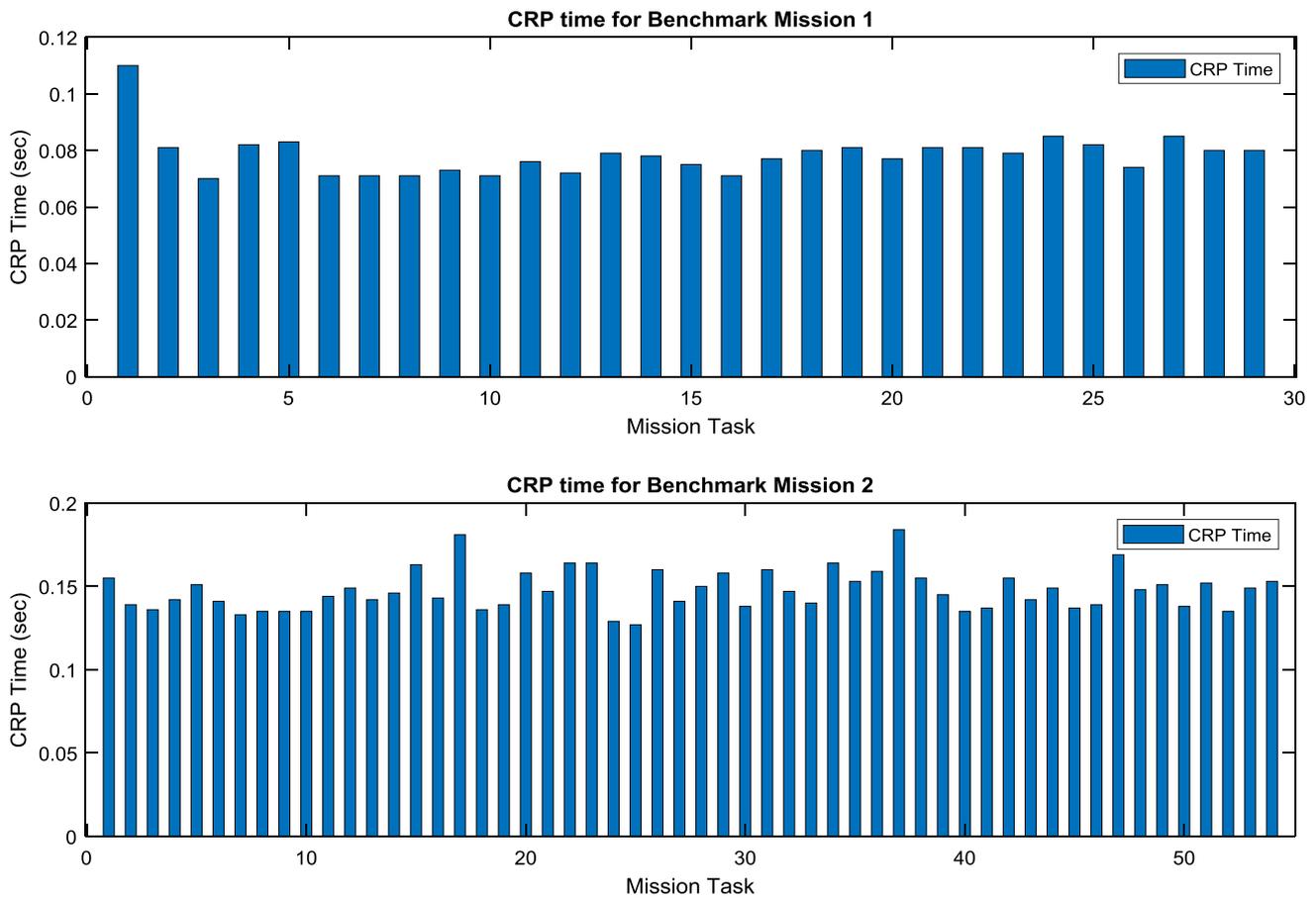


Fig. 17 UAV successfully delivering its cargo



**Fig. 18** CRP time of benchmark missions

- There is no re-learning of complex statistical reasoning models or networks whenever the knowledge or evidence changes because, in our approach, potential solutions are evaluated in real-time and a statistical model for reasoning is generated in real-time by the MEP.
- Autonomous behaviour can be controlled more effectively because in our approach, the probabilities and marginal probabilities provided by the AEFQ algorithm enables a finer control of the statistical fitness evaluations of the state-transitions.
- The high-level control provided by the CRP is more representative of human cognition, because in our approach, the OWA is followed. This means the action of a state-transition may be less probable, but not impossible. This gives the CRP powerful reasoning capabilities.
- The fitness of a state-transition is a true representation of the environment because, the MEP applied in our approach, guarantees an accurate probability assignment, based only on the constraint averages derived from the mission constraints and environmental evi-

dence. There are no other subjective control parameters or bias in the fitness quantification.

### 6.3.2 Time efficiency

The objective of this study is the real-time, high-level control provided by the CRP. Therefore, the time efficiency of the CRP, i.e. the time taken by the AE-SPSO to find an optimal solution for a mission task, is evaluated. Optimization algorithms, including the PSO algorithm, is known for the extensive time it takes to converge on an optimum. This is especially true for large, multi-dimensional and real search spaces. However, in our approach, the search space is finite and discrete, allowing the AE-SPSO to find optimal solutions in acceptable and sufficient time. Moreover, the control parameters of the AE-SPSO makes it easy to scale the performance of the PSO when the search space increases.

Figure 18 shows the time the CRP took to find an optimal solution for each of the tasks of each mission.

The average CRP time for benchmark mission 1 was 0.0785 s and for benchmark mission 2, the average CRP time was 0.1477 s. These times were found to be

completely suitable for the high-level control of the UAV, while executing its missions.

The behaviour of the UAV in both benchmark missions is demonstrated in the accompanying videos [35, 36].

### 6.3.3 Simulation constraints

The performance of the UAV may appear slow in the videos. This is because the complex integration architecture of the AirSim simulator and the Unity games engine is not optimal and causes a considerable time lag between the simulator and the games engine. It was observed that, at high velocity, the UAV would overshoot its target destination in the Unity games engine. This resulted in the target position parameters reported by the AirSim to be inconsistent from that reported by the games engine. This caused the UAV to wrongly interpret its position and therefore miss its objectives.

To improve the performance, a delay was explicitly implemented between the execution of mission tasks, in order to give the games engine and simulator time to synchronise. Assisted by the explicit delay, the UAV would autonomously correct its positioning, by repeating the task (see Figs. 8 and 13), while constantly reducing its velocity according to the fitness of the task. At low velocity, the positioning of the UAV was quite accurate and it could achieve its objectives. With the autonomous velocity control, the UAV was able to successfully reach the charging station in benchmark mission 1 and was able to successfully collect and deliver its cargo in benchmark mission 2. It should be noted that this is a simulation problem which, is unlikely to occur in a real-world scenario.

## 7 Conclusion and future work

In real-world scenarios, semi-autonomous systems, such as exploratory robots, operate in environments which may constantly change. Therefore, it must be trivial and computationally inexpensive to alter a robot's behaviour, by updating its KB and/or mission objectives in real-time. This is especially important for remotely deployed robotic systems, such as extra-terrestrial exploration robots, where communication time and bandwidth are at a premium.

In this research study, an approach which combines expert knowledge and a cognitive reasoning process was introduced. The approach simplifies the management of the knowledgebase by domain experts, while providing the system with autonomous reasoning, by optimizing the knowledge, given the real-time environmental knowledge. The approach presented here introduces a simple knowledgebase structure, which is easy to maintain and less error-prone. The results of the research also show that,

the robot can successfully execute its missions by optimizing the expert-provided knowledge and dynamically and progressively generating and executing a high-level controller.

Further study could extend this approach by investigating multi-objective knowledge optimization in order to generate parallel FSA's with specific sub-task objectives. This will be useful for high-level control of a robotic system with multiple capabilities. For example, a FSA for flight-control, a FSA for camera control and a FSA for gripper control.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## References

1. Van Harmelen FL, Lifschitz V, Porter B (2008) Handbook of knowledge representation, 1st edn. Elsevier, Amsterdam
2. Baars BJG, Nicole M (2012) Fundamentals of cognitive neuroscience—a beginner's guide. Elsevier, Amsterdam
3. Antsaklis PJ, Rahnema A (2018) Control and machine intelligence for system autonomy. *J Intell Robot Syst* 91:23–34
4. Perico DH, Homem TPD, Almeida AC, Silva IJ, Vilão CO, Ferreira VN, Bianchi RAC (2018) Humanoid robot framework for research on cognitive robotics. *J Control Autom Electr Syst* 29:470–479
5. Hernández García D, Monje CA, Balaguer C (2017) Task oriented control of a humanoid robot through the implementation of a cognitive architecture. *J Intell Robot Syst* 85:3–25
6. Schiffer S (2016) Integrating qualitative reasoning and human-robot interaction in domestic service robotics. *KI Künstliche Intelligenz* 30:257–265
7. Drenjanac D, Tomic SDK, Kuhn E (2015) A semantic framework for modeling adaptive autonomy in task allocation in robotic fleets. In: 2015 IEEE 24th international conference on enabling technologies: infrastructure for collaborative enterprises (WET-ICE), pp 15–20
8. Martínez-Tenor A, Fernández-Madrigal JA, Cruz-Martín A, González-Jiménez J (2018) Towards a common implementation of reinforcement learning for multiple robotic tasks. *Expert Syst Appl* 100:246–259
9. Hong A, Igharoro O, Liu Y, Niroui F, Nejat G, Benhabib B (2018) Investigating human-robot teams for learning-based semi-autonomous control in urban search and rescue environments. *J Intell Robot Syst* 94:669–686
10. Sampedro C, Rodríguez-Ramos A, Bavle H, Carrio A, de la Puente P, Campoy P (2018) A fully-autonomous aerial robot for search and rescue applications in indoor environments using learning-based techniques. *J Intell Robot Syst* 95:601–627
11. Rishwaraj G, Ponnambalam SG (2017) Integrated trust based control system for multirobot systems: development and experimentation in real environment. *Expert Syst Appl* 86:177–189
12. Getoor L, Grant J (2006) PRL: a probabilistic relational language. *Mach Learn* 62:7–31
13. Muggleton S, De Raedt L (1994) Inductive logic programming: theory and methods. *J Log Program* 19:629–679

14. Van Laer W, Dehaspe L (1994) Applications of a logical discovery engine. In: Proceedings of the AAAI workshop on knowledge discovery in databases, pp 263–274
15. Wellman MP, Breese JS, Goldman RP (1992) From knowledge bases to decision models. *Knowl Eng Rev* 7:35–53
16. Wang DZ, Chen Y, Goldberg S, Grant C, Li K (2012) Automatic knowledge base construction using probabilistic extraction, deductive reasoning, and human feedback. In: Presented at the proceedings of the joint workshop on automatic knowledge base construction and web-scale knowledge extraction, Montreal, Canada, 2012
17. Quinlan JR (1990) Learning logical definitions from relations. *Mach Learn* 5:239–266
18. Luis MVJ, Holguin GA, Mauricio HL (2018) A methodology for movement planning in autonomous systems with multiple agents. In: 2018 IEEE 2nd Colombian conference on robotics and automation (CCRA), 2018, pp 1–6
19. Shoukry Y, Nuzzo P, Balkan A, Saha I, Sangiovanni-Vincentelli AL, Seshia SA, Pappas GJ, Tabuada P (2017) Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming. In: 2017 IEEE 56th annual conference on decision and control (CDC), 2017, pp 1132–1137
20. Kamil F, Hong TS, Khaksar W, Moghrabiah MY, Zulkifli N, Ahmad SA (2017) New robot navigation algorithm for arbitrary unknown dynamic environments based on future prediction and priority behavior. *Expert Syst Appl* 86:274–291
21. Tenorio-González AC, Morales EF (2018) Automatic discovery of concepts and actions. *Expert Syst Appl* 92:192–205
22. Meyer P-J, Dimarogonas DV (2019) Hierarchical decomposition of LTL synthesis problem for nonlinear control systems. *IEEE Trans Autom Control* 64(11):4676–4683. <https://doi.org/10.1109/TAC.2019.2902643>
23. Bellman R (1957) A markovian decision process. *Indiana Univ Math J* 6:679–684
24. Rodriguez-Ramos A, Sampedro C, Bavle H, de la Puente P, Campoy P (2019) A deep reinforcement learning strategy for UAV autonomous landing on a moving platform. *J Intell Robot Syst* 93:351–366
25. Low ES, Ong P, Cheah KC (2019) Solving the optimal path planning of a mobile robot using improved Q-learning. *Rob Auton Syst* 115:143–161
26. Brass S, Lipeck UW (1989) Specifying closed world assumptions for logic databases. In: Demetrovics J, Thalheim B (eds) *MFDBS 89: 2nd symposium on mathematical fundamentals of database systems Visegrád, Hungary, June 26–30, 1989 proceedings*. Springer, Berlin, pp 68–84
27. Biba M (2009) Integrating logic and statistics—novel algorithms in markov logic networks. *VDM Verlag Dr Muller Aktiengesellschaft & Co. KG, Saarbrücken*
28. Das PK, Sahoo BM, Behera HS, Vashisht S (2016) An improved particle swarm optimization for multi-robot path planning. In: 2016 international conference on innovation and challenges in cyber security (ICICCS-INBUSH), 2016, pp 97–106
29. Cai Q, Long T, Wang Z, Wen Y, Kou J (2016) Multiple paths planning for UAVs using particle swarm optimization with sequential niche technique. *Chin Control Decis Conf (CCDC)* 2016:4730–4734
30. Walha C, Bezine H, Alimi AM (2013) A multi-objective particle swarm optimization approach to robotic grasping. In: 2013 international conference on individual and collective behaviors in robotics (ICBR), 2013, pp 120–125
31. Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science, 1995. *MHS'95*, pp 39–43
32. Yuhui S, Eberhart R (1998) A modified particle swarm optimizer. In: The 1998 IEEE international conference on evolutionary computation proceedings, 1998. *IEEE world congress on computational intelligence*, 1998, pp 69–73
33. Wei-Neng C, Jun Z, Chung HSH, Wen-Liang Z, Wei-gang W, Yu-Hui S (2010) A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE Trans Evol Comput* 14:278–300
34. Blower DJ (2013) Information processing—the maximum entropy principle volume two: createspace independent publishing platform, 2013
35. De Jager D (2019) UAV benchmark mission 1 [Video (mp4)]. <https://youtu.be/pBZD1yOH19E>
36. De Jager D (2019) UAV benchmark mission 2 [Video (mp4)]. [https://youtu.be/JV\\_f9GDWTSU](https://youtu.be/JV_f9GDWTSU)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.