Self-Reconfiguring Robotic Framework Using Fuzzy and Ontological Decision Making

Affan Shaukat, Guy Burroughes and Yang Gao

Abstract Advanced automation requires complex robotic systems that are susceptible to mechanical, software and sensory failures. While bespoke solutions exist to avoid such situations, there is a requirement to develop generic robotic framework that can allow autonomous recovery from anomalous conditions through hardware or software reconfiguration. This paper presents a novel robotic architecture that combines fuzzy reasoning with ontology-based deliberative decision making to enable self-reconfigurability within a complex robotic system architecture. The fuzzy reasoning module incorporates multiple types of fuzzy inference models that passively monitor the constituent sub-systems for any anomalous changes. A response is generated in retrospect of this monitoring process that is sent to an Ontology-based rational agent in order to perform system reconfiguration. A reconfiguration routine is generated to maintain optimal performance within such complex architectures. The current research work will apply the proposed framework to the problem of autonomous visual navigation of unmanned ground vehicles. An increase in system performance is observed every time a reconfiguration routine is triggered. Experimental analysis is carried out using real-world data, concluding that the proposed system concept gives superior performance against non-reconfigurable robotic frameworks.

1 Introduction

Robotics and complex autonomous systems are becoming increasingly pervasive in almost every field of life [6, 17]. The evolution of robotic research and real-world application domains has resulted in a great number of sophisticated robotic

A. Shaukat (✉) · G. Burroughes · Y. Gao
Faculty of Engineering and Physical Sciences, Surrey Space Centre,
University of Surrey, Guildford GU2 7XH, UK
e-mail: a.shaukat@surrey.ac.uk

G. Burroughes
e-mail: g.burroughes@surrey.ac.uk

Y. Gao
e-mail: yang.gao@surrey.ac.uk

Y. Bi et al. (eds.), *Intelligent Systems and Applications*,
Studies in Computational Intelligence 650, DOI 10.1007/978-3-319-33386-1_7

architectures that are prone to mechanical, software and sensory failures. Although manual servicing is regularly carried out, it can become extremely time-consuming, expensive, risky [17] and even infeasible (such as for Martian rovers) in certain situations. As such there is a requirement to develop generic architectures of robotic systems that can enable self-reconfiguration of critical application software or hardware units in response to any environmental perturbation, performance deterioration and system malfunction. Hardware-based reconfiguration may include physical sensors and manipulators while software-based reconfiguration would allow the use of the most optimal software programs that is appropriate for a given situation. Rational agent-based architectures perform reconfiguration of higher-level goals, actions and plans [**?**]. A summary of the different types of reconfiguration that may take place in a robotic architecture is presented in Fig. 1.

This article addresses the problem of reconfiguration in complex robotic systems by using a multi-layered framework that builds upon the model introduced in [16]. The architecture presented in [16] incorporates the agent as a part of the reconfiguration layer. The proposed model adopts the rational agent as a separate layer, hence extending it to a three-layered architecture, thus further allowing the reconfiguration of higher level goals, actions and plans (for example multi-agent systems). This work will seek to establish the advantage of the proposed reconfigurable robotic framework by performing a number of quantitative and qualitative assessments.

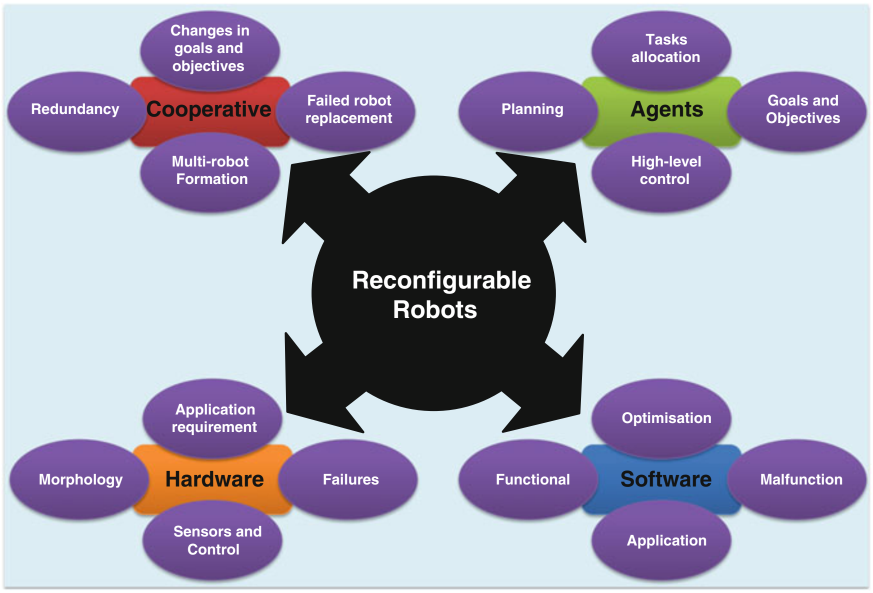


Fig. 1 Types of reconfiguration in robotic frameworks

2 Background

2.1 Reconfigurable Systems

In literature, the field of self-reconfigurable robotic systems has traditionally focused on the design and development of techniques and algorithms that can process reconfiguration of the morphological characteristics of modular systems [21]. However, there is an increasing understanding that the concept of reconfiguration in autonomous systems and robots expands beyond variation in geometry, shape, and hardware composition [12]. Many contemporary robotic architectures follow an isolated engineered system approach in comparison to many contemporary autonomous systems, which employ a *hybrid* agent architecture with many of the hardware and software sub-systems supervised by a *Rational Agent* [5]. Their highly modular and distributed architectures enable them with software and hardware reconfiguration within their capabilities in response to novel situations; whereas, the agent makes *rational* high-level decisions, define goals and perform planning/re-planning (refer to Fig. 1) [5]. While these advantages have yet to be fully utilised in the robotics field [21], there are a number of relevant examples in literature that have sought to develop techniques that can enable reconfiguration with limitations according to their specific applications.

Systems that can perform reconfiguration of controllers based on errors or specific performance requirements have also been proposed literature, known as *Plug & Play* control, and the controllers described as polymorphic [5]. In software architectures, there are examples of self-managing distributed processing systems, which can adapt to stochastic changes such as; *Autonomic Computing* [7], *Autonomic Control Loops* [5], *Self-CHOP, and Self-Organising Component Based Software* [18], MAPE-K [5] etc. For a few agent-based complex robotic systems, a number of multi-agent based reconfigurable architectures have also been proposed in literature [2]; however, certain limitations do exist in terms of generality.

2.2 Fuzzy Inference Systems

Fuzzy inference systems (FIS) are used in robotic systems to incorporate intelligent reasoning in situations where physical sensors and actuators have an intrinsic notion of uncertainty. Such systems have been widely used for practical control architectures. FIS have also proven to be very useful in logic-based decision-making and discerning patterns from low-level sensory inputs [16] and top-down reasoning in multi-layered hierarchical architectures that incorporate machine vision and learning [14, 20]. FIS use logic applicable to fuzzy sets with degrees of membership, which are formulated in terms of membership functions valued in the real unit interval [0, 1]. These membership functions can either be singular values, intervals or even sets of intervals. Machine learning techniques (such as, *neural networks*) are commonly used for optimisation and development of these membership functions. However, such methods rely on the scale of training data and the number of adjusted parameters [16]; therefore, they may not be suitable for complex robotic architectures with multiple levels of data abstraction and rational decision making. To avoid such problems, membership function definitions and parameter setting can also be performed using empirical knowledge of the robot sensory and control settings. Such knowledge can be extracted using *Ontology*.

2.3 Ontology

For a system to perform reconfiguration autonomously the system must be self-aware. In other words, the system must have a usable System knowledge representation of its internal subsystems, if it is going to be able attempt any autonomic characteristics. To accomplish this a Ontological model can be used.

An Ontology is a formal description of the concepts and relationships that can formally exist in a domain. Ontologies are often equated with taxonomic hierarchies of classes, class definitions, and the subsumption relation, but ontologies are not limited to these forms. An Ontology can be described as a Description Logic built on top of a Description Logic, where a Description Logic is any decidable fragment of FOL, that models concepts, roles and individuals, and their relationships [11].

The most prominent robot ontology research is the newly formed IEEE-RAS working group entitled Ontologies for Robotics and Automation (ORA) [13]; however, nothing is concrete as of yet. The goal of this working group is to develop a standard ontology and associated methodology for knowledge representation and reasoning in robotics and automation, together with the representation of concepts in an initial set of application domains.

Another example of an emerging robotics domain ontology is Intelligent Systems Ontology [13]. The purpose of this ontology is to develop a common, implementation-independent, extensible knowledge source for researchers and developers in the intelligent vehicle community. It provides a standard set of domain concepts, along with their attributes and inter-relations allow for knowledge capture and reuse, facilitate systems specification, design, and integration. It is based on a service based description akin to web service's WSDL [4], hence the use of a OWL-S [10] as the base language. It similarities to WSDL are profound, but what it does have is an ability to describe complex agents. What it does not have, is an ability to describe the physical environment; however, other ontologies can be appended to this ontology to make a more domain complete ontology.

3 A Reconfigurable Robotic Framework

The proposed multi-layered self-reconfigurable robotic architecture is illustrated in Fig. 2. It is a three-layered architecture; an *Agent*, a *Reconfiguration*, and an *Application* layer.

Briefly:

- The *Application Layer* contains the traditional elements of a robotic architecture, and doesn't contain any of the elements required for self-reconfiguration.

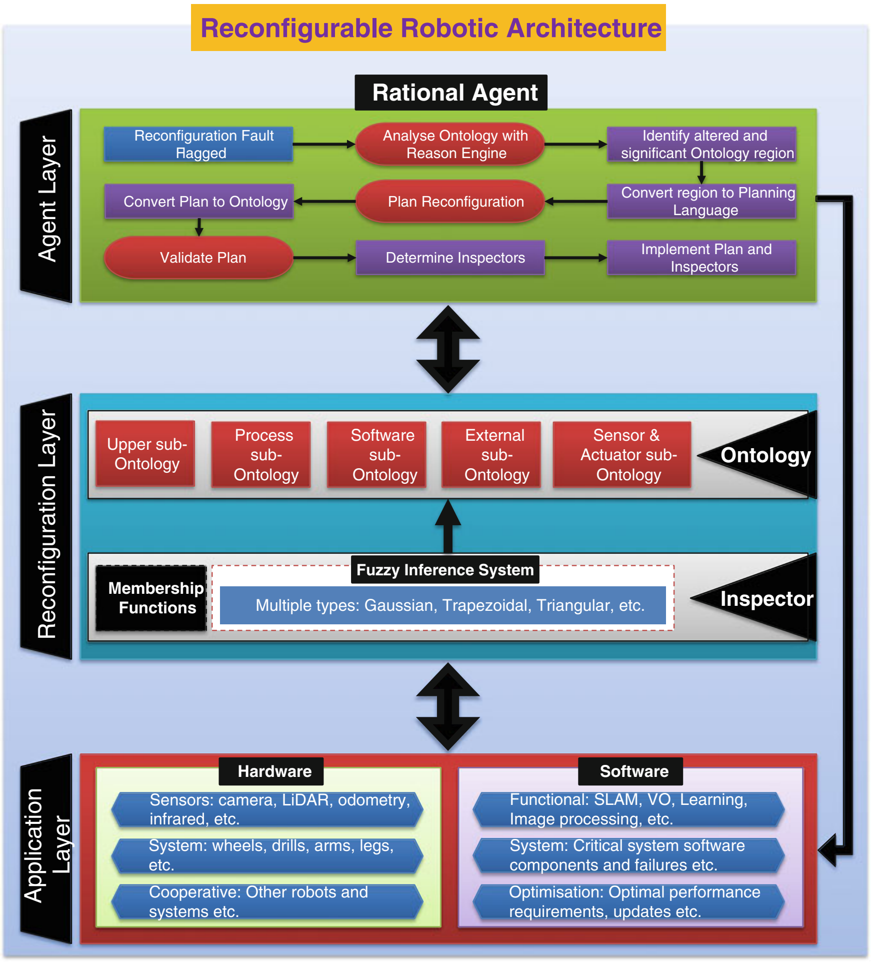

Reconfigurable Robotic Architecture
Rational Agent
Agent Layer
Reconfiguration Fault Ragged
Analyse Ontology with Reason Engine
Identify altered and significant Ontology region
Convert Plan to Ontology
Plan Reconfiguration
Convert region to Planning Language
Validate Plan
Determine Inspectors
Implement Plan and Inspectors
Reconfiguration Layer
Upper sub-Ontology
Process sub-Ontology
Software sub-Ontology
External sub-Ontology
Sensor & Actuator sub-Ontology
Ontology
Membership Functions
Fuzzy Inference System
Multiple types: Gaussian, Trapezoidal, Triangular, etc.
Inspector
Application Layer
Hardware
Sensors: camera, LiDAR, odometry, infrared, etc.
System: wheels, drills, arms, legs, etc.
Cooperative: Other robots and systems etc.
Software
Functional: SLAM, VO, Learning, Image processing, etc.
System: Critical system software components and failures etc.
Optimisation: Optimal performance requirements, updates etc.

Fig. 2 Proposed self-reconfigurable robotic architecture

- The *Reconfiguration Layer* contains the elements of the system required for self-monitoring and self-awareness, and thus monitors the Application Layer's performance.
- The *Agent Layer* contains the Agent that computes Self-Reconfiguration for the system using the knowledge in the Reconfiguration Layer.

The layer of abstraction are ordered in this manner to: maximise modularity, which in turn allows the layers to be adapted and changed more easily; not require the Application Layer components to be integrated deeply with reconfiguration elements, which limits the number of faults/errors that Reconfiguration and Agent Layer unintentionally inject to the Application Layer; and it allows the Application Layer to be easily divorced from the Reconfiguration elements if they are no longer desired.

The Frameworks basic operations breaks into three control loops. The first control loop is the standard robotics control loops, which occurs in the Application Layer, this maybe multiple nested control loops like any non-self-reconfigurable robotic framework. The next control loop is in the Reconfiguration Layer, this separate control loop monitors the application layer, updating its internal knowledge of the system. The Reconfiguration Layer's control loop, if required, triggers the next control loop. The Final Control loop is the Agent control loop, which occurs in the Agent Layer. In the Agent control loop the required reconfiguration is calculated.

The Reconfiguration Control loop can carrying on monitoring system, while the Agent Layer is computing. If new information is pertinent to the Agent control loop, the Agent Control loop may be restarted. When the Reconfiguration control loop triggers the Agent Layer, it puts the Application into a safe mode. Safe mode is a operation mode, which limits the amount of damage a system can do itself, whilst performing sub-optimally. It is analogous to the system Safety modes of operations in robotic space-craft.

The following is a series of more detailed discussions in to these three layers.

3.1 Application Layer

The Application layer contains all the hardware and software components that exist in most other autonomous systems or robotic architectures. This layer contains all the classic elements of a non-reconfigurable robotics architecture. This layer contains no elements necessary for self-reconfiguration.

The hardware components may be:

- ***Sensors***: are various types of exteroceptive and interoceptive sensors such as, Cameras, LiDAR, Touch Sensors, Encoders, GPS, etc. For example, Sensor Reconfiguration may involve the process of replacing one type of sensor with another due to malfunction, fallible inputs or precision requirements from the higher level reconfiguration layer.
- ***System***: components are adherent to the system architecture and form an intrinsic part of some mechanism, such as, wheels, grippers, manipulators, etc. In this case,

reconfiguration may be required by the system course of actions, for example, orientation of the wheels to perform a specific type of manoeuvre.
- ***Cooperative scenarios***: are situations consisting of multiple robots working towards common mission goals, where a reconfiguration may be required when one of the robots either stops working, or the rational agent generates a new plan that may require an alternative formation of the robots.

Similarly, the software components can be:

- ***Functional components***: are application software and programs that use sensor data to enable autonomous capabilities, such as visual navigation from camera images, map generation and localisation from LiDAR, etc. Most of these algorithms are intrinsically stochastic, and their performance can be affected by spurious data. In such cases specific software can be reconfigured with more robust algorithms with better or more optimal performance for the scenario presented.
- ***System***: components are critical system software units that run and execute application software, for example, operating systems, boot-loaders, safety monitoring systems, etc. Any failure in this category can cause a critical paralysis of the whole system; however, a reconfigurable architecture can avoid this well in advance if certain anomalies are detected by the higher levels.

Under normal conditions, the system would operate without any anomalies and the application layer performance may not significantly differ from that of an engineered robotic system. However, within this framework, any changes that occur in the application layer are monitored and analysed. Any anomalies that are identified can then be dealt with as deemed appropriate by the Reconfiguration Layer.

3.2 Reconfiguration Layer

The Reconfiguration layer operates at a higher-level than the Application layer within the hierarchy of abstraction, which contains the necessary elements for building a knowledge of the changes in the Application layer and enabling the Rational Agent to trigger a reconfiguration. This from an Autonomic point of view would be the Self-Aware, Self-Monitoring, and Self-Analysing element of the system.

The principal components of this layer are: the Inspector, and the Ontology. The following are two individual discussions into these principal components.

3.2.1 Inspector

An Inspector is the lowest layer in the hierarchy of the Reconfiguration Layer, which passively monitors the Application Layer for *Reconfiguration Anomalies*, via FIS. The Inspector monitors the Application Layer for Reconfiguration Anomalies.

Reconfiguration Anomalies are things that deviate from what is standard, normal, or expected, which may affect an application optimal operation. These anomalies can be defined as:

A ***Self-Reconfiguration System Failure*** (SR-Failure) is an event that occurs when the configuration deviates from optimal strategy. These are distinct and adjoint from standard software failures, because a non-optimal configuration is not the same as the Fault.

A ***Self-Reconfiguration System Error*** (SR-Error) is that part of the system state that may cause a subsequent SR-Failure: a SR-Failure occurs when an SR-Error reaches the service interface and alters the service.

A ***Self-Reconfiguration System Fault*** (SR-Fault) is the adjudged or hypothesized cause of an SR-Error.

SR-Faults can include environmental, software, and hardware changes. Furthermore, these definitions limit SR-Faults to individual components. Whilst, SR-Failures can propagate through a system with these definitions, a SR-Fault is their point or points of origin.

To monitor for SR-Faults, outputs from the application layer are translated into fuzzy confidence values that provides a *semantic* notion of performance to the *Ontology*. To date, the proposed framework has implemented fuzzy inference based on *Mamdani's* method [9] due to its simple structure of *min–max* operations and high degree of success rate in many complex control architectures. However other types can be easily integrated in the Inspector. This allows for generic monitoring of low-level systems within the proposed framework, making it appropriate for any type of complex robotic system.

The FIS system uses multiple fuzzy variables that associate confidence values with the inputs from the Application layer; for example, these can be *errors* and *processing time* of an *extended kalman filter* within a simultaneous localisation and mapping system [1]. FIS subsumes conjunctive operators (T-norms; min, prod etc.) and disjunctive operators (T-conorms; max, sum, etc.) as well as hybrid operators (combinations of the previous operators). In particular, the T-norm (triangular norm) is generally a continuous function, $[0, 1] \times [0, 1] \Rightarrow [0, 1]$. Membership function definition depends upon the appropriate modelling parameters of the various units in the Application layer, and they can either be *gaussian*, *trapezoids* or *triangular*. A fuzzy rule-base combines the input variables in order to compute the rule strength. The antecedents within the fuzzy rule-base use the fuzzy operators to compute a single membership value for each rule. The outputs from multiple rules in the fuzzy rule-base are aggregated into a single fuzzy set, which is generally performed using the *max* operator. The fuzzy logic inference system allows the use of multiple input fuzzy variables, for example *processing time* and *error* to infer decisions as the consequence of the a priori set of fuzzy logic rules. The final output from the fuzzy inference system is a single confidence value instead that is used by the Ontology.

Consider an example of a fuzzy variable associated with the computation time characterised as,

$$compute\_time = \langle \alpha, \mathbb{U}_\alpha, \mathbb{R}(\alpha) \rangle, \tag{1}$$

where α is the computation time required for processing within the Application layer, $\mathbb{U}_\alpha$ is the universe of discourse, and $\mathbb{R}(\alpha)$ is the fuzzy membership of α. Two piecewise linear continuous fuzzy membership functions, i.e., 'comp_time$_{\text{low}}(\alpha)$' and 'comp_time$_{\text{high}}(\alpha)$', can be associated with α, as defined in [16].

Similarly, a fuzzy variable associated with the estimation error can be characterised as follows,

$$estimation\_error = \langle \beta, \mathbb{U}_\beta, \mathbb{R}(\beta) \rangle, \tag{2}$$

where β the error in state estimation of a localisation technique, $\mathbb{U}_\beta$ is the universe of discourse, and $\mathbb{R}(\beta)$ is the fuzzy membership of β. Similar to the previous case, two piecewise linear continuous fuzzy membership functions, i.e., 'post_error$_{\text{low}}(\beta)$' and 'post_error$_{\text{high}}(\beta)$', are associated with the input error from the Navigation system β [16].

The output variable defines the performance of the system in the Application layer on the *aggregated* confidence value computed from an inference technique applied to the fuzzy input variables,

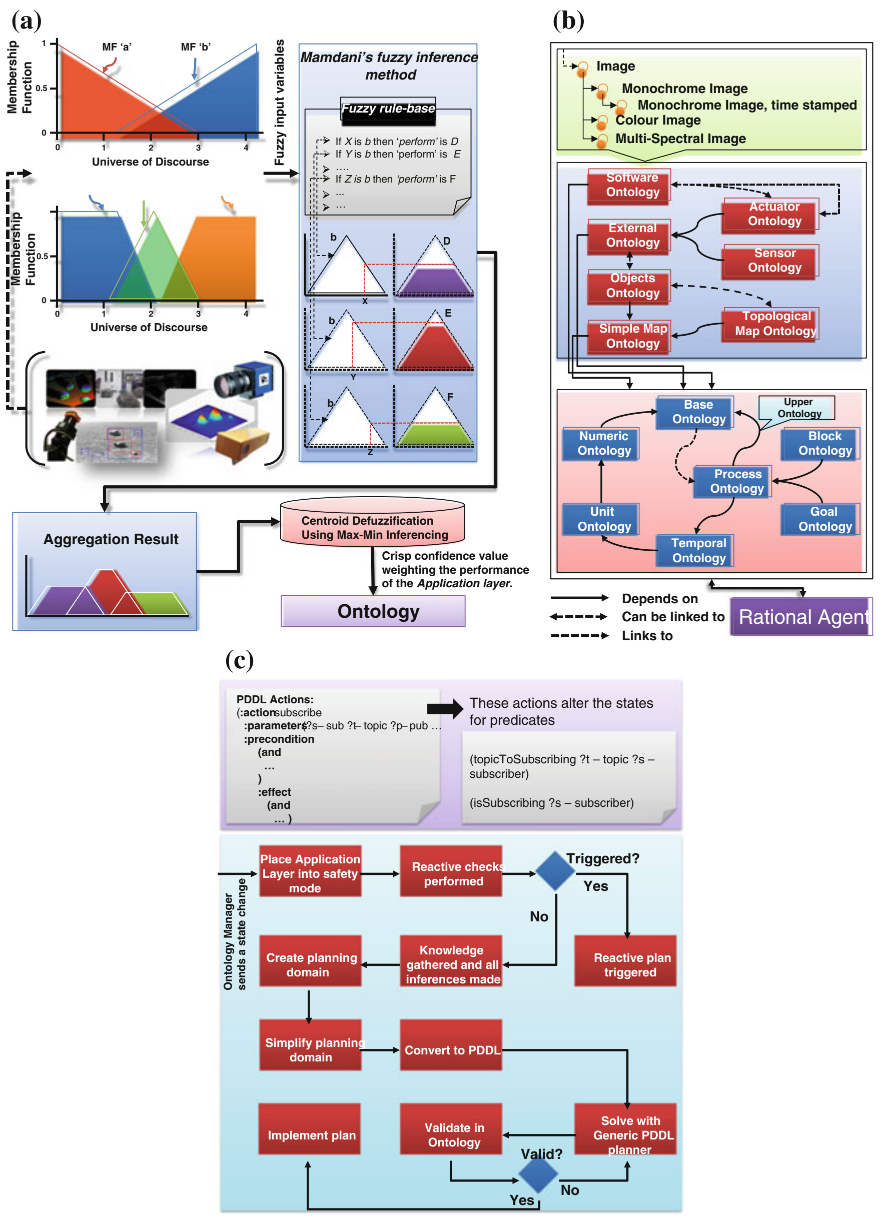
$$system\_performance = \langle \gamma, \mathbb{U}_\gamma, \mathbb{R}(\gamma) \rangle. \tag{3}$$

Two piecewise fuzzy membership functions, i.e., 'system_performance$_{\text{bad}}(\gamma)$' and 'system_performance$_{\text{good}}(\gamma)$' associated with the system performance output provide a confidence measure on how "*good*" or "*bad*" the lower-level SLAM system is performing [16].

A fuzzy rule-base combines the fuzzy input variables to compute the rule strength. The antecedents within the fuzzy rule-base use the fuzzy operators *AND* as t-norm and *OR* as t-conorm in order to compute a single membership value for each rule, for example:

– *if* comp_time$_{\text{low}}(\alpha)$ *AND* post_error$_{\text{low}}(\beta)$ *THEN* system_performance$_{\text{good}}(\gamma)$,
– *if* comp_time$_{\text{high}}(\alpha)$ *OR* post_error$_{\text{high}}(\beta)$ *THEN* system_performance$_{\text{bad}}(\gamma)$,

An implication operator is used to truncate the consequent's membership function output. The outputs from multiple rules within the fuzzy rule-base are aggregated into a single fuzzy set (refer to Fig. 3a for a graphical representation of the process). The final output is a single value, which is the final stage of the inference process known as *defuzzification* [14, 16, 20].



```
PDDL Actions:
(:action subscribe
  :parameters (?s – sub ?t – topic ?p – pub …
  :precondition
    (and
      …
    )
    :effect
      (and
       … )
```

```
(topicToSubscribing ?t – topic ?s – subscriber)

(isSubscribing ?s – subscriber)
```

Fig. 3 Illustrations of the FIS (**a**), the Ontology (**b**) and the Rational Agent (**c**)

3.2.2 Ontology

The Ontology is the upper layer in the hierarchy of the reconfiguration layer, which contains all information used for a successful self-reconfiguration of the robotic system. The Ontology has been designed as a Modular Ontology with an Upper Ontology. This splitting of the Ontology into different modules increases extensibility, usability and readability of the Ontology. Furthermore, modularity adds the ability to perform validity checking and inference gathering on sub-ontologies, because there are fewer assertions and instances, it is more efficient.

The primary modules of the Ontology are:

- *Upper Ontology* forms the base syntax and logic for the description of goals and capabilities of modules, and is further sub-divided into essential modules like numerics, timings etc.
- *Process sub-ontology* describes the processes of autonomous software
- *Software sub-ontology* describes inner working and inter-communications of autonomous software
- *External sub-ontology* describes external environments
- *Sensor and Actuator sub-ontology* are both self-explanatory, and are dependent on the software and external sub-ontology

The Capabilities of Software and Hardware shall be represented in an extended IOPE format. The Input and Output (IO) are the communication input channels. Each channel connects to their appropriate service grounding, having a communication type, message type, rates, and their communications properties, they also have their necessity defined (i.e., is the communication input necessary for completion). The message formats and communication type exist in a subsumption hierarchy. The subsumption hierarchy has increasingly complex and specific properties and logics. The Precondition and Effect describe, in an extensible subsumptive description logic, the precondition for a service and the effect of implementing stated service. These conditions affect both the Environment and internal conditions. Then, there are non-functional properties, that can be used for quality of service metrics. Additionally, most configurations are encoded in states, so that each configuration represents a different IOPE. Continuous configurations parameters are encoded to states via parametric equations, they should be used not to make large changes, but mere parameter changes (e.g. rate). If continuous configuration parameter do make large changes to the functional operation of a service, it should be abstracted to another discontinuous parameter and another continuous parameter.

The Process logic is encoded as a single control loop of the autonomous software, multiple control loops can be described as a hierarchy, similar to a Hierarchical task network. The non-functional environmental and internal parameter condition change based on their separate conditions. For example, the battery usage for a service is given as a conservative estimate for complete standard cycle loop, battery usage is then verified at the lower abstraction layer, like the standard action planner and scheduler. The process is then logically modelled by a reduced version of Computational Tree Logic (CTL) translated to First order logic [19].

3.3 Agent Layer

The (*Rational*) Agent layer analyses the Ontology and performs planning/re-planning, followed by the *Validation* and *Verification* process, and finally implementation. The Rational Agent is the main source of triggering a self-reconfiguration routine in the lower layers. The basic process of the Rational Agent is outlined in the system diagram, Fig. 3c. The self-reconfiguration control loop is started by the *Inspector* in the *Reconfiguration layer* every time the system develops a fault or detects an anomalous behaviour.

The Rational agent triggers reconfiguration. The Rational agent takes a tiered view of the world. A subsumptive three-tier model is applied to the planning problem to reduce complexity and consequently reduce individual planning subspaces, and rendering the process of action planning more efficient [8]. Similarly, the planning problem for self-reconfiguration is not choreographing every action of the system, but the self-reconfiguration system adds another level to the subsumptive architecture that deals with a reconfiguration of the lower levels. Thus, the planning problem is to find a configuration of the system which allows the lower levels to perform their intended action.

The Rational agent has both reactive and deliberative elements, and is initialised by a change in the Ontology. Once the Rational agent has been informed by the Ontology Manager about a change, it will go through a series of steps in the attempt to rectify or optimise the system. For the purposes of speed, initial steps of the Rational agent are all reactive. The Reactive component has pre-calculated criteria, which when triggered react with a corresponding pre-calculated plan. The pre-calculated plans and criteria are prepared by a deliberative fault (or world-change) injection analysis, which calculates the most likely changes to the system. If none of the reactive criteria are fulfilled the Rational agent will use its deliberative techniques to devise an alternate plan.

To deal with the intrinsic complexity the Rational agent uses many techniques to mitigate the inherent complexity to remain efficient. The Deliberative component will first place the application layer in a *safety mode*, which halts all future operations until the safety of the system can be evaluated by the Rational agent. The Deliberative component then gathers all relevant information to the reconfiguration process and convert it into a form which can be used to generate the reconfiguration plan. While this could be achieved via first order logic in the Ontology's inference engine, directed planning algorithms with heuristics are more suited for this long chain style planning problem. Thus, the planning problem is converted into a generic planning language (i.e., PDDL2.1), then a generic Planner is used to find a self-reconfiguration plan. Using generic Planners allows for fast optimisation of planner algorithm and heuristic choice. This plan is then verified in the Ontology and implemented by the Rational agent.

The planning problem in the case of the self-reconfiguration of the application layer has an initial state that includes the (initial) world state. The goal of the planning problem includes the final world state. The possible actions in the planning domain are the services and the connection of services in which the appropriate communication link is used. Services follow the form of input, output, precondition, and effect (IOPE). The input and output refer to their communication requirements (e.g. publishing rate), and the precondition and effect refer to systems world state. The precondition and effect may include energy requirements, context requirements, other world state changes, and service requirements, e.g. whether a camera has previously been connected to the application layer. For all parameter configurations of a service there are different defined actions. The service can have discontinuous and continuous attributes and parameters. The services are assumed to perform with the most conservative scenario in respect to usable resources. This allows the other three levels of the hierarchical execution structure to be confident in being able complete their goals. The service can have any number of logical preconditions and restrictions on them. Then the solution to this planning problem is a set of services, their parameter configurations, and their connections.

To generate the PDDL, first order logic rules are used, which allows for very high level of extensibility since new rules for PDDL creation can be introduced easily. To minimise the planning domain sent to the PDDL planner, multiple space size reduction techniques are used. Using the knowledge that the plan will primarily focus on connecting services and then service configurations, a planning domain can be pruned for unreachable services that would not be apparent to generic algorithms as efficiently.

4 Experimental Scenarios

To demonstrate and prove the robustness of the Self-Reconfiguring Robotic Framework against unforeseen changes in the system, this paper presents two experiments incorporating software and hardware based self-reconfiguration scenarios. In the following, these individual scenarios will be discussed with their respective results followed by a discussion into the significance of these experimental findings.

The experimental workstation used for running the FIS comprises a quad-core Intel(R) Xeon(R) X5482 CPU (3.20 GHz) running Linux (Ubuntu 12.04, 64-bit architecture). Workstation used for the Agent, Application Layer, and the Ontology also had similar specifications.

4.1 SLAM in Unstructured Environments

The aim of the experiment is to test the ability of the Self-Reconfiguring Robotic Framework to optimise the system on the basis of changing system performance, which can alter because of changing environments and anomalous operations of the Application layer. The scenario in this experiment is a Planetary rover using a monocular camera based navigation system. The navigation system implemented in the Application Layer is the PM-SLAM localisation system introduced in [1].

PM-SLAM (Planetary monocular simultaneous localization and mapping) is a highly modular, monocular SLAM system for use in planetary exploration. PM-SLAM's modular structure allows multiple techniques to be implemented, such as, distinct SLAM filters, and visual feature tracking techniques. For example, multiple types of vision-based features may be implemented, SURF features or hybrid of semantic blob-based [1, 15] feature tracking, both having their own performance characteristics for different environments.

A visual SLAM system is susceptible to accumulated posterior state estimation error, and accruing computation time over multiple iterations [1]. The Self-Reconfiguring Robotic Framework attempts to maintain an optimal level of computational load and error over the course of operation, through self-reconfiguration of the subcomponents of PM-SLAM. For the purpose of monitoring the Application layer, this paper implements a FIS based Inspector. This Inspector monitors the predicted localisation error and computation loop time to generate a confidence value that quantifies the performance of the localisation system. The predicted localisation error for this experiment is derived from the covariances estimated by the SLAM filter. This Confidence measure is used to update the Ontology, which is then used by the Agent Layer to compute, and if necessary implement a new optimal configuration of the Application Layer.

A subset of the images generated by the European Space Agency field trial in the Atacama Desert, Chile in 2012 using the SEEKER rover platform is used for this experimental scenario [1].

4.1.1 Experimental Setup

PM-SLAM and the other Application Layer modules have been implemented in C++ using ROS and OpenCV. The Agent Layer is implemented in JAVA using Apache's JENA. The reactive component of the Agent Layer has been not implemented for this test, as it does not aid in demonstrating the capabilities and efficiency of the Agent Layer. The Ontology is implemented in OWL (Web Ontology Language), as it is the most widely used and supported Ontology Language, with a wealth of mature tools like Protégé and Apache JENA. The FIS Inspector has been implemented in MATLAB. Both the Agent Layer and the Ontology have been implemented as generic systems such that they can be adapted for various other types of robotic applications.

4.1.2 Experimental Results

The application layer takes two different configurations during the experimental process. An initial configuration of the application layer utilising SURF-based feature tracking, a depth perception module, and an EKF SLAM filter and a reconfigured application layer after the FIS Inspector confidence has dropped below acceptable levels (as determined by the Rational Agent). This configuration utilises a hybrid semantic blob and SURF feature tracking technique and an EKF SLAM filter [1].

As discussed in Sect. 3.2.1, the FIS generates confidence values using the inputs (*error* and *computation time*) from the SLAM system, which are added to the Ontology and used by the Rational agent to examine the system performance over the course of operation. The tests are run on-line over the images from the dataset; however, a reconfiguration routine is triggered by the Rational agent responding to the change in the Ontology as soon as the system performance confidence value goes below a specified threshold (0.5 in the current case). Referring to Fig. 4b, a significant reduction in the computation time is observed for the proposed reconfigurable system. In contrast, the engineered system continues its operation without any reconfiguration and therefore the computation time continues to increase. Similarly, Fig. 4c presents the behaviour of the system's posterior error over multiple iterations, which is marginally increased following the reconfiguration. This is because of the sudden change in the type and cardinality of the visual features used by the SLAM filter. However, this is deemed by the system as a suitable compromise to maximise the over performance of the system. This is also observed for the FIS-based system performance measure as shown in Fig. 4a, where the confidence value increases following the reconfiguration in the proposed system, while it continues to decrease for the engineered system.

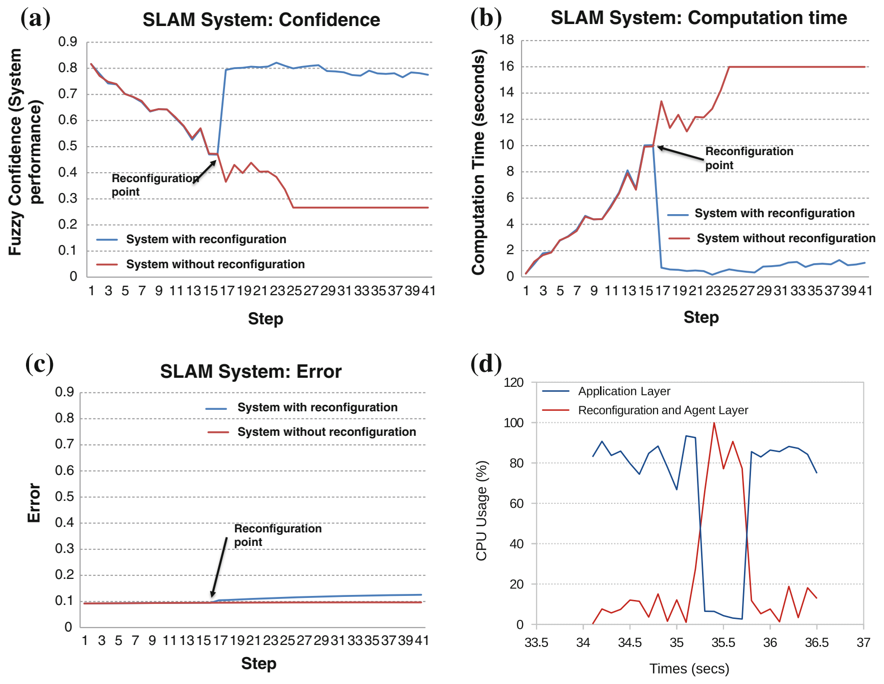


Fig. 4 FIS performance-based confidence output (**a**), computation time (**b**), weighted error (**c**) and CPU usage (**d**) for the proposed self-reconfigurable system against an engineered system in the SLAM scenario

As Fig. 4d shows the reconfiguration process takes approximately 0.46 s. The average slam loop time before the reconfiguration is approximately 5.65 s; therefore, the reconfiguration time is comparatively fast compared to the applications layer process. Similarly, the computation usage of the Application layer and Reconfiguration Layer, when they are in active full use, is of a similar amount. Moreover, the computational usage of the reconfiguration layer whilst not in actively calculating a reconfiguration plan is comparatively minimal.

4.1.3 Discussion

From the results presented in Sect. 4.1.2, it is clear that the Self-Reconfiguring Robotic Framework is capable of optimising the Application Layer, in that an improvement in performance is observed following a reconfiguration, as compared to a standard engineered system that does not allow for any reconfiguration. In particular, it demonstrate the system ability to reconfigure complex autonomous software at run-time, while maintaining continuity of service. Furthermore, it is clear that the system can self-optimise a system in such a way that includes compromising other systems, to maximise the systems overall performance. Moreover, the Self-Reconfiguring Framework demonstrates an ability to perform reconfiguration relatively efficiently.

4.2 Hardware in Complex Systems

The aim of this experiment is to test the ability of the Self-Reconfiguring Robotic Framework to respond to degrading or faulty hardware, to improve the continuity of operations of the system and optimise the systems overall performance. Similar, to the previous experiment a rover navigation system will be the scenario, with a camera that will be artificial impaired as to simulate degrading hardware or environmental conditions that may affect camera lens and cause unwanted artefacts in the input images. A common factor for noisy visual inputs is granulated dust particles. Such type of noisy conditions can be simulated by adding "*salt-and-pepper*" noise to images [1], hence simulating problem with one of the camera hardware. The Self-Reconfiguring Robotic Framework is expected to maintain continuity in the system, optimising the Application Layer, and eventually switching to another camera entirely.

Similar to the last system the Application Layer is monitored by a FIS based Inspector, which uses *entropy* as a statistical measure of randomness that can characterise the texture of the input image, and computation loop time to generate a confidence value for the performance of the Application layer system. This value is used to update the Ontology, which is then used by the Agent Layer to compute, and if necessary implement a new optimal configuration of the application layer.

In order to prove that the Self-Reconfiguring Robotic Framework can cope with such hardware problems, the framework should demonstrate a reconfiguration at run-time to provide continuity of service, even after a significant event in contrary to a traditionally engineered system, which may suffer a catastrophic failure.

4.2.1 Experimental Setup

The experimental setup is the same as for the previous scenario in Sect. 4.1.1. However, unlike the previous scenario the second camera (right) in the stereo pair from the SEEKER dataset, is used to simulate an auxiliary camera. Test images from left and right cameras are subsampled by adding *"salt-and-pepper"* noise over a varying scale of 0–100 % with a step size of 1 for the left camera (simulating very fast deterioration), and a step size of 0.1 for the right camera (simulating negligible damage).

4.2.2 Experimental Results

The application layer takes two different configurations as in the previous experiment, i.e., the initial configuration where the left camera is initially used by subsystem, followed by a reconfiguration process triggered by the Rational Agent switching the subsystem to the alternative right camera, which is still in working condition compared to the left camera.

Referring to Fig. 5, the Self-Reconfiguring Robotic Framework is capable of optimising the Application Layer, by switching to an alternative camera as soon as the fuzzy confidence measure goes below the specified threshold. This results in an improvement in performance of the system. Comparative analysis in this figure shows that a standard engineered system without any reconfiguration continues to use images from the faulty camera.

The FIS generated confidence values depend upon the entropy value and *computation time* from the SLAM subsystem, which are added to the Ontology and used by the Rational agent to check system performance. The tests are run on-line over the images from the sub-sampled dataset with simulated noise. A reconfiguration is performed by the Rational agent responding to the change in the Ontology on the basis of the system performance confidence value. Referring to Fig. 5b, the computation time is significantly reduced, while an improvement is observed in the entropy measure. This is a result of switching the faulty camera (left camera) to an alternative one that is still in working order (right camera). The engineered system continues its operation without any reconfiguration and therefore the computation time and the entropy measure continue to deteriorate over time. Similarly, the FIS-based system performance measure as shown in Fig. 5a, increases following the reconfiguration in the proposed system, while it continues to decrease without any reconfiguration.

As Fig. 5d shows the reconfiguration process takes approximately 0.47 s. The average slam loop time before the reconfiguration is approximately 3.61 s; therefore,

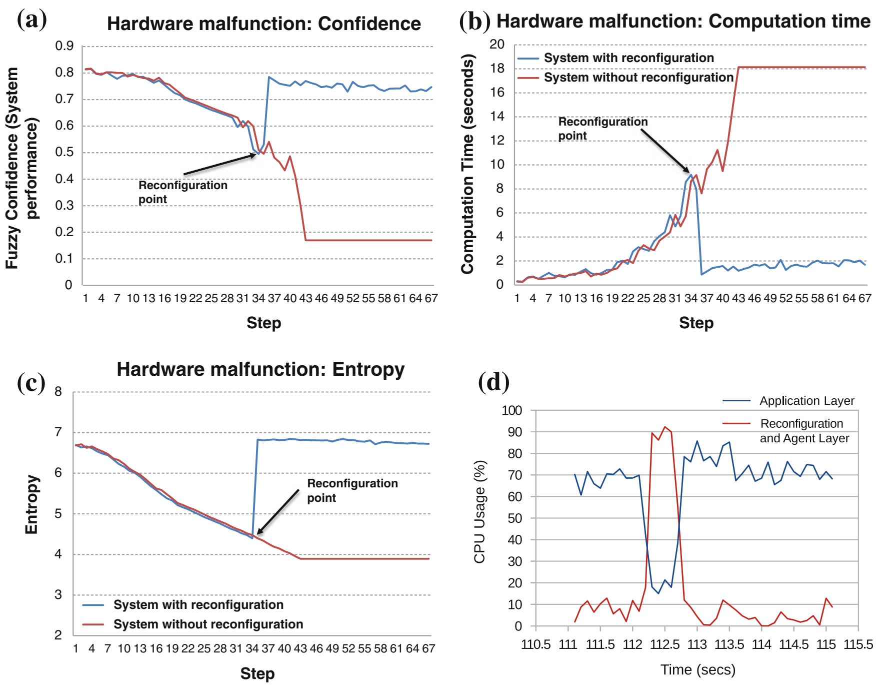


Fig. 5 FIS performance-based confidence output (**a**), computation time (**b**), entropy (**c**) and CPU usage (**d**) for the proposed self-reconfigurable system against an engineered system in the faulty hardware scenario

the reconfiguration time is comparatively fast compared to the applications layer process. The average time post reconfiguration is 1.80 s, which is comparatively slow compared to the Reconfiguration process. Similarly, the computation usage of the Application layer and Reconfiguration Layer, when they are in active full use, is of a similar amount. Moreover, the computational usage of the reconfiguration layer whilst not actively calculating a reconfiguration plan is comparatively minimal.

4.2.3 Discussion

From the results presented in Sect. 4.2.2, it is clear that the Self-Reconfiguring Robotic Framework is capable of optimising the Application Layer, in that an improvement in performance is observed following a reconfiguration, as compared to a standard engineered system that does not allow for any reconfiguration. In particular, the framework can reconfigure hardware for autonomous systems at run-time.

Furthermore, it demonstrates the system ability to cope with inferred measures of the systems performance. Moreover, the Self-Reconfiguring Framework demonstrates an ability to perform reconfiguration relatively efficiently.

5 Conclusion

This paper introduced a Self-reconfiguring Robotic framework in the form of a hierarchy combining fuzzy reasoning with ontology-based deliberative decision making to enable self-reconfigurability within a complex robotic system and increase the fault tolerance and robustness to unforeseen changes in the system and the environment. The framework is designed to be generic and therefore allows to be implemented in a wide variety of robotic applications. An Ontological model is used for system knowledge representation to enable it to be self-aware of its constituent subsystems. A fuzzy inference system quantifies the performance of the lower-level subcomponents within the Application layer in terms of fuzzy confidence measures which are intuitive for human operators that may be monitoring the system. At the highest level of the system hierarchy; a rational-agent constantly checks the underlying levels of the system for anomalies or inconsistent behaviour that can potentially cause fatal damage to itself, harm to the environment or failure to achieve desired objectives and trigger a reconfiguration in the Application layer.

Two different experimental setups were used to test the proposed system and its advantages against a system that does not allow reconfiguration in any situation. The setups involved real-world robotic hardware, such as, autonomous rovers and vision sensors that were used to generate the test and analysis data. The outcome of these experiments established that the proposed Self-Reconfiguring Robotic Framework either maintains or improves the performance of the over all system in challenging operating conditions. The layered design of the system, has proven to be an effective method for abstracting the reconfiguration of a complex system, without significantly influencing the underlying system (i.e., application layer) during normal operations.

In future work, an extension to the work presented in this paper could include inspectors that monitor the application layer using other logical inferencing techniques to discover various performance characteristics such as software faults for reconfiguration.

Acknowledgments The work presented here was supported by the U.K. Engineering and Physical Sciences Research Council under Grant EP/J011916/1 [Reconfigurable Autonomy] and Grant EP/I000992/1 [MILES]. The authors are very grateful to the Autonomous Systems Group, RAL Space, for providing the SEEKER dataset used for the experiments. No new data was created in the study.

References

1. Bajpai, A., Burroughes, G., Shaukat, A., Gao, Y.: Planetary monocular simultaneous localization and mapping. J. Field Robot. **33**(2), 229–242 (2015)
2. Bojinov, H., Casal, A., Hogg, T.: Multiagent control of self-reconfigurable robots. Artif. Intell. **142**(2), 99–120 (2002). International Conference on MultiAgent Systems 2000
3. Guy, B., Yang, G.: Ontology-based self-reconfiguring guidance, navigation, and control for planetary rovers. J. Aeros. Inform. Syst. doi:10.2514/1.I010378 (2016)

4. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., et al.: Web services description language (wsdl) 1.1 (2001)
5. Dennis, L.A., Fisher, M., Aitken, J.M., Veres, S.M., Gao, Y., Shaukat, A., Burroughes, G.: Reconfigurable autonomy. KI-Künstliche Intelligenz **28**(3), 199–207 (2014)
6. Gao, Y., Frame, T.E.D., Pitcher, C.: Peircing the extraterrestrial surface: integrated robotic drill for planetary exploration. Robot. Autom. Mag. IEEE **22**(1), 45–53 (2015)
7. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing—degrees, models, and applications. ACM Comput. Surv. **40**(3), 7:1–7:28 (2008)
8. Knight, R., Fisher, F., Estlin, T., Engelhardt, B., Chien, S.: Balancing deliberation and reaction, planning and execution for space robotic applications. In: Proceedings. 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001, vol. 4, pp. 2131–2139. IEEE (2001)
9. Mamdani, E.H., Assilian, S.: An experiment in linguistic synthesis with a fuzzy logic controller. Int. J. Hum.-Comput. Stud. **51**(2), 135–147 (1999)
10. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al.: Owl-s: Semantic markup for web services. W3C member submission, 22:2007–04 (2004)
11. McGuinness, D.L., Van Harmelen, F., et al.: Owl web ontology language overview. W3C Recomm. **10**(10), 2004 (2004)
12. Rouff, C., Hinchey, M., Rash, J., Truszkowski, W., Sterritt, R.: Autonomicity of nasa missions. In: Proceedings Second International Conference on Autonomic Computing, 2005. ICAC 2005, pp. 387–388. IEEE (2005)
13. Schlenoff, C., Prestes, E., Madhavan, R., Goncalves, P., Li, H., Balakirsky, S., Kramer, T., Miguelanez, E.: An ieee standard ontology for robotics and automation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012, pp. 1337–1342 (2012)
14. Shaukat, A., Gilbert, A., Windridge, D., Bowden, R.: Meeting in the middle: a top-down and bottom-up approach to detect pedestrians. In: 21st International Conference on Pattern Recognition (ICPR), 2012, pp. 874–877 (2012)
15. Shaukat, A., Spiteri, C., Gao, Y., Al-Milli, S., Bajpai, A.: Quasi-thematic features detection and tracking for future rover long-distance autonomous navigation. In: 12th Symposium on Advanced Space Technologies in Robotics and Automation, Noordwijk, the Netherlands, European Space Agency, ESTEC (2013)
16. Shaukat, A., Burroughes, G., Gao, Y.: Self-reconfigurable robotics architecture utilising fuzzy and deliberative reasoning. SAI Intell. Syst. Conf. **2015**, 258–266 (2015)
17. Shaukat, A., Gao, Y., Kuo, J.A., Bowen, B.A., Mort, P.E.: Visual classification of waste material for nuclear decommissioning. Robot. Auton. Syst. **75**, Part B, 365–378 (2016)
18. Sterritt, R., Hinchey, M.: Spaace iv: self-properties for an autonomous & autonomic computing environment—part iv a newish hope. In: Seventh IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASe), 2010, pp. 119–125 (2010)
19. Vakili, A., Day, N.A.: Reducing ctl-live model checking to first-order logic validity checking. In: Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design, pp. 215–218. FMCAD Inc (2014)
20. Windridge, D., Felsberg, M., Shaukat, A.: A framework for hierarchical perception-action learning utilizing fuzzy reasoning. IEEE Trans. Cybern. **43**(1), 155–169 (2013)
21. Yim, M., Shen, W.-M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., Chirikjian, G.S.: Modular self-reconfigurable robot systems [grand challenges of robotics]. IEEE Robot. Autom. Mag. **14**(1), 43–52 (2007)