Degradation, Adaptation and Self-Reconfiguration of Autonomous Robots

Sandor M. Veres, Jonathan M. Aitken, and Owen McAree

Abstract—A theoretical framework and associated algorithmic theory is presented for the interlinked issues of robotic hardware degradations, adaptations and self-reconfiguration of autonomous robots with respect to their performance specifications. Performance specification is formalised through a simplified and jointed modal logics that is specific for this problem area. In terms of the physical environments surrounding the robot, a constrained set of hybrid systems descriptions is used to characterise the environmental assumptions made for the satisfaction of specifications. This formulation permits the representation of a variety of environmental circumstances and dynamical changes that a robot can face during its service. Building on logic based specifications, grounding is represented for a mathematical model and theory for an autonomous robot's degraded performance. As degradation is to be accompanied by self adaptation and reconfiguration of robots, a theory of optimal redundancy is developed to support self-reconfiguration. This is followed by proposals of effective techniques of distributed computation for fast reconfiguration and system adaptation. It is illustrated that the presented theory can be implemented in ROS or a similar middleware.

Index Terms—Autonomous robots, degradation, reconfigurable control, adaptation, ROS middleware, computer-aided control system design.

I. INTRODUCTION

The design, deployment and maintenance of any technical system is carefully managed by engineers. As systems become more complex, engineers incorporate automation and autonomous decision capability into the operation of systems. The aim of autonomous operation is to simplify the operation of a technical system without the need for close human supervision. Apart from reduced human workload, additional benefits of autonomous operations are a possible increase in performance or in efficiency [1], [2], [3]. However, without careful design and good grounding, the introduction of automation can make the operation of such systems more complex or even dangerous, thereby cancelling out its benefits, as highlighted in the Überlingen mid air collision [4].

First of all, designers need clarity and ultimately a formal approach to specifying the properties of a robotic or automated machine system. Performance specification needs to be made against a varying environment, where the robot or machine (simply referred to as ‘robot’) is deployed. Even the best robots have their limitations and hence the user manuals, provided by robot manufacturers, need to describe the limitations of the robot’s use and applicability and define the appropriate circumstances when and where it can be deployed. For instance, for flying robots the strength of wind/storm and humidity can be limiting factors, for ground robots there can be limits to the unevenness of the terrain where the robot is appropriate and qualified to be used.

A. Robot Specifications

A simple theory of specifications will be defined here through interval-temporal-epistemic-deontic (iTED) logic as adopted from [5], [6], [7]. The semantics of this logical system can be defined by an ordered set of possible Kripke worlds [8] and an infinite set of hybrid systems to represent the possible physical worlds, where performance-relevant properties are expressed in terms of temporal logic of sensing, actions and modal logic of obligations, permissions and knowledge of other agents’ awareness and possible goals. These specification issues are addressed in terms of a combined use of temporal, deontic and epistemic logic introduced here for formalising requirements on robot behaviours. Permissions are needed to enable freedom of the agent to solve problems under varying circumstances without unnecessary limitations. Obligations of keeping to behaviour rules is a most important requirement of advanced robots for practical use. Reasoning about other agent’s knowledge is needed so that robots can be effective in cooperation to avoid unnecessary communication with humans and other robots. Finally, temporal logic is fundamentally needed as the world is modelled by observations during a time period. Using only current time (present tense) logic leads to inefficiencies of logical decision making and increases the complexity of verifying the correctness of robotic decision making software. Goals are achieved during a time period and a good number of behaviour rules contain concepts of past, current and future time. Temporal logic also allows us the formulation of time-constrained requirements of performance. If this logic is compatible with robot reasoning then the robot can make decisions to satisfy its performance requirements in terms of interval temporal logic [5], [9] that we adopt for our combined theory of specification, adaptation, degradation and self-reconfiguration.

B. Degradation

We will show that robot capabilities can be made to form a partially ordered set in terms of comparable levels of performance. The robots hardware or software can degrade in time for various reasons and this changes a robot’s capabilities through limiting its sensing and movements. The nature of the environment may not change but the deteriorating sensing
and actuation means that the robot needs to handle changing dynamics in its external interactions. We will present a performance degradation theory in terms of possible changing physical and computational conditions of the robot. A theory of changes in the Boolean evaluations of robot capabilities is developed. The main result is our derivation of a tree graph of robot capabilities and our analysis of its basic properties.

C. Reconfiguration

The problem of autonomous reconfiguration is fundamental in developing intelligent autonomous processes, machines and robots with a broad range of capabilities. As these machines become more complex, so do their problems of reconfiguration, especially in case if some of their modules fail and immediate action is needed to maintain performance. It is important for the future of intelligent machines that proper understanding of the nature of degradation is developed that is addressed by self-reconfiguration of complex autonomous engineering systems.

This paper sets out a framework that can provide a formal grounding for reconfigurable automation in systems based on their task and device dependency graphs. This will focus on developing a generic mathematical model of reconfiguration that is readily analysable for its effectiveness. It substantially expands on the work in [10] by adding time dependency into the model of the network. By specifying the component availability the system is then able to select a configuration pattern required to meet the current conditions, without needing to undergo a lengthy reset process during non-interruptible operation. The theory presented can be applied to robots through the use of Robot Operating System (ROS) [11].

The idea of using partially ordered sets of deteriorations originates from Morse et al. [12], [13], which is developed here into a fuller mathematical theory to clarify a few key points not discussed in [12], [13].

D. Paper Outline

Section 2 introduces a limited but decidable modal logic that is based on interval-temporal, epistemic and deontic logic (iTED) to enable both the formulation of requirements on a robots behaviour as well as to allow the robot to test whether its pqst actions or plans would satisfy those requirements. Section 3 builds on iTED to formalise a partially ordered set of degradations and proves some of its basic properties. Section 4 provides formal definitions of dependency relationships of specification formulae that is much needed for the build up of efficient realtime reconfiguration of the software or hardware of a robot. The combined theory is completed in Section 5-6-7 by the presentation of realtime reconfiguration theory and algorithms when ROS-like middleware is used to implement the software of the robot. Finally, Section 8 presents a practical example of the combined theory proposed in the paper.

II. Modal logics of specifications

In this section we define a robot’s logic $R_L$ as a temporal deontic logic extended with epistemic capabilities. For compatibility of expressions, and for the robot’s ability to assess whether it satisfies requirements, the same $R_L$ is used for a specification language to define performance specifications and behaviour rules. Such a logic enables designers as well as the robot to reason in terms of obligations and permissions in the physical world. In addition, $R_L$ is extended by epistemic logic expressions such as that “robot B knows of event E and B is permitted to take action A within the next hour” or the “human H knows of event E”. $R_L$ is to satisfy the following requirements of expressivity and predicate abstractions.

1) Current time predicates with arguments are to express: abstractions of sensing, communication events and of actions taken by the robot in the environment. These three groups of predicates include all possible interactions with the environment.

2) Temporal logic. The time series of current time predicates can be used to infer temporal logic statements by the robot and for the robot in specifications. Temporal logic expressions are needed in specifications to be able to express past and current circumstances and future events or actions, which the robotic agent needs to do or disallowed to do within some time period.

3) Epistemic logic based specifications are needed to be able to express what the robot is aware of and what it knows other agents are aware of. This is needed in requirements in human robot interactions and in cooperation of robot teams. Requirements of cooperativeness can be formulated by the use of formulae in epistemic logic.

4) Deontic logic formulae are able to define what the robot is obliged to do and to what it is allowed to do. Apart from its behaviour constraints, the robot has well defined freedom to achieve its obligations through its permissions in deontic logic.

A. Modal logic notations

This section summarises the formalism used in our joint temporal-epistemic-deontic (iTED) logic proposed for a formal specifications of robot behaviour. Table I displays

**Examples of iTED logic notations for a UAV:** Current predicates, which express sensory events, communications or actions by the robot are simple statements like

\[
\text{InIdleMode, InHoverModeAtHeight(h), LandAtLocation(L), etc.}
\]

which can be augmented by time or a time interval as, after their current occurrence, they may not be valid in the future and what remains valid is that they occurred at some past time instant, for instance at some past time instant $t$: \[\Box(t) \text{ InIdleMode}, \Box(t) \text{ InHoverModeAtHeight(h)}, \Box(t) \text{ LandingAtLocation(L)}, ...\]. Few of these sensed or acted events last only for an instant and they are better associated with an interval of time as \[\Box([t_1, t_2]) \text{ InIdleMode}, \Box([t_1, t_2]) \text{ InHoverModeAtHeight(h)}, \Box([t_1, t_2]) \text{ LandingAtLocation(L)}, ...\]. Which, for instance, imply that \[\Box(t) \text{ InHoverModeAtHeight(h)}\] holds for all $t \in [t_1, t_2]$. To economise on memory storage, robots can be made to record the longest periods for which a

<table>
<thead>
<tr>
<th>Condition</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always in the future:</td>
<td>□</td>
<td>sometime: 0</td>
</tr>
<tr>
<td>During future period:</td>
<td>□ ((t, t_2))</td>
<td>sometime in period: 0(t, t_2)</td>
</tr>
<tr>
<td>Always in the past:</td>
<td>◻</td>
<td>sometime past: 0(t, t_2)</td>
</tr>
<tr>
<td>During past period:</td>
<td>◻ ((t, t_2))</td>
<td>sometime past period: 0(t, t_2)</td>
</tr>
<tr>
<td>At a past time:</td>
<td>◻</td>
<td>sometime ago: 0(t)</td>
</tr>
<tr>
<td>Next:</td>
<td>⊗</td>
<td>next: ◻</td>
</tr>
<tr>
<td>Only next:</td>
<td>⊗</td>
<td>only next: ⊗</td>
</tr>
<tr>
<td>Just past:</td>
<td>⋄</td>
<td>just past: ⋄</td>
</tr>
<tr>
<td>Permitted:</td>
<td>♦</td>
<td>permitted that: ♦</td>
</tr>
<tr>
<td>Obligatory:</td>
<td>◻</td>
<td>obligatory that: ◻</td>
</tr>
<tr>
<td>Forbidden:</td>
<td>⌂</td>
<td>forbidden: ⌂</td>
</tr>
<tr>
<td>Exits:</td>
<td>⊥</td>
<td>exists: ⊥</td>
</tr>
<tr>
<td>Proves:</td>
<td>⊢</td>
<td>proves: ⊢</td>
</tr>
<tr>
<td>Implies:</td>
<td>⊢</td>
<td>implies: ⊢</td>
</tr>
<tr>
<td>For all:</td>
<td>∀</td>
<td>or: ∀</td>
</tr>
<tr>
<td>And:</td>
<td>∧</td>
<td>and: ∧</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Condition</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicate had been valid, i.e. overlapping ones are joined into one. For specifications of robot performance, one can also apply temporal operators referring to the future, for instance:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>□(t)∀L( TakingOffAtLocation(L_1)) \rightarrow</td>
<td>symbolises that, whilst airborne, a UAV should maintain a separation distance (d_{\text{ENV}}) from an object (O_{\text{ENV}}), when there is no plan to inspect the object (P^I(O_{\text{ENV}})), greater than the safe distance, (d_{\text{safe}}), typically 50m from people and properties not under direct control, defined as objects of interest in the environment (O_{\text{ENV}}). If the separation falls below (d_{\text{safe}}) it should always move away from the object.</td>
</tr>
<tr>
<td></td>
<td>∀L2((t + 200, t + 1800)) LandingAtLoc(L_2) \land d(L_1, L_2) &lt; 700)</td>
<td>symbolises that, whilst airborne, a UAV should not move closer or be further from an object, (O_{\text{ENV}}), than a set inspection distance, (d_{\text{inspection}}), when there exists a plan to inspect the object, (P^I(O_{\text{ENV}})). If this is the case a controller should be used to maintain the distance, (\text{maintainDistance}(O_{\text{ENV}}, d_{\text{inspection}})).</td>
</tr>
<tr>
<td></td>
<td>(□(t, t + T)(B \geq B_{\text{crit}}) \land ∃L2((t, t + T)) LandingAtLoc(L_2)))</td>
<td>symbolises that a UAV must and obey the airspace restrictions in place within the current region of flight, (R_{\text{max}}), this means that Prohibited, Restricted or Danger Areas should be avoided, as should areas with temporary restrictions. These temporary restrictions should be updated pre-flight as they are published at short notice through the Notice to Airmen (NOTAM) system and regularly through Aeronautical Information Circulaires.</td>
</tr>
<tr>
<td></td>
<td>□(t)∀L( TakingOffAtLocation(L_1)) \rightarrow</td>
<td>symbolises that, whilst airborne, a UAV should not move closer or be further from an object, (O_{\text{ENV}}), than a set inspection distance, (d_{\text{inspection}}), when there exists a plan to inspect the object, (P^I(O_{\text{ENV}})). If this is the case a controller should be used to maintain the distance, (\text{maintainDistance}(O_{\text{ENV}}, d_{\text{inspection}})).</td>
</tr>
<tr>
<td></td>
<td>□(t)∀L( TakingOffAtLocation(L_1)) \rightarrow</td>
<td>symbolises that a UAV should always move away from the aircraft.</td>
</tr>
<tr>
<td></td>
<td>□(t)∀L( TakingOffAtLocation(L_1) \land \Diamond\text{Inpect}(O_{\text{ENV}})) \rightarrow</td>
<td>symbolises that a UAV must and obey the airspace restrictions in place within the current region of flight, (R_{\text{max}}), this means that Prohibited, Restricted or Danger Areas should be avoided, as should areas with temporary restrictions. These temporary restrictions should be updated pre-flight as they are published at short notice through the Notice to Airmen (NOTAM) system and regularly through Aeronautical Information Circulaires.</td>
</tr>
<tr>
<td></td>
<td>□(t)∀L( TakingOffAtLocation(L_1)) \rightarrow</td>
<td>symbolises that a UAV should always move away from the aircraft.</td>
</tr>
<tr>
<td></td>
<td>□(t)∀L( TakingOffAtLocation(L_1)) \land \Diamond\text{Inpect}(O_{\text{ENV}}) \rightarrow</td>
<td>symbolises that a UAV should always move away from the aircraft.</td>
</tr>
<tr>
<td></td>
<td>□(t)∀L( TakingOffAtLocation(L_1)) \land \Diamond\text{Inpect}(O_{\text{ENV}}) \rightarrow</td>
<td>symbolises that a UAV should always move away from the aircraft.</td>
</tr>
</tbody>
</table>

The logical operators defined in this section can be used to specify robot behaviour in combination with predicates with arguments over domains that typically contain members of a world model. For example:

\[ \Box(t)\forall L(\text{TakingOffAtLocation}(L_1)) \rightarrow \Box(t, t + T)(B \geq B_{\text{crit}}) \land \exists L2(t, t + T)\text{LandingAtLoc}(L_2) \]

symbolises that a UAV, taking off from a location \(L_1\), should always land on a landing site \(L_2\) with battery level \((B)\) no lower than critical level \(B_{\text{crit}}\) (typically 20%).

\[ \Box(t)\forall L(\text{TakingOffAtLocation}(L_1)) \land \Diamond\text{Inpect}(O_{\text{ENV}}) \rightarrow \Box\Diamond\text{moveAwayFrom}(O_{\text{ENV}}) \]

symbolises that, whilst airborne, a UAV should maintain a separation distance \(d_{\text{AC}}\), from an aircraft \(O_{\text{AC}}\), greater than the safe distance, \(d_{\text{safe}}\). If the separation falls below \(d_{\text{safe}}\), it should always move away from the aircraft.
and environmental specifications. Depending on these hybrid system set-based definitions, some specification formulae can be satisfied or not satisfied.

B. Robot reasoning in iTED

Automated reasoning is needed for robots to self-verify that their planned physical actions in space and time will not break some of their human set behaviour constraints while achieving some goals. Here we examine whether robot reasoning can be kept within an iTED calculus that is decidable in the following sense.

Assume that at any given time \( t \) the following set of iTED predicates and formulae are available.

- Current sensory predicates \( S_t \) and temporal logic formulae \( P_t \), about the past experience and operational states of the robot.
- An evolving "foresight" world model \( M_{i,T} \) computed by the robot for a time horizon \( T \) at a dense sequence of time instances.
- A movement plan of the robot \( \Pi(t, T) \) for some time horizon \( T \) produced by its continuous space/time replanner.
- An abstraction engine \( A \) that generates a set of future predicates \( F_{i,T} \) from the foresight model \( M_{i,T} \) and movement plan \( \Pi(t, T) \).

**Definition 1.** Let \( L \) be a class of iTED formulae with notations as in Table I. \( L \) is called decidable for time horizon \( T \) if there is an algorithm that is guaranteed decide within finite time whether \( S_t, F_{i,T} \models \eta \) for any \( \eta \in L \). If there is a universal bounded time period to verify the validity of formulae in \( L \) then \( L \) is called strongly decidable.

Note that \( F_{i,T} \) is dependent on \( \Pi(t, T) \) and decidability does not imply satisfiability of requirements by some suitable plan of the robot. It only means that the robot is able to decide whether its planned actions will violate any of its performance requirements. Guaranteeing the existence of a plan, which does not violate any of the requirements, is not implied by decidability.

For practical purposes we do not need the full expressive power of iTED. We are interested instead to identify a subclass of iTED that is decidable and therefore can help the robot to ensure that it plans and acts within its behaviour specification. The following is a constrained iTED, which is a subset of iTED to provide decidability:

**Definition 2.** iTED\(_p\) is a set of past tense logic formulae using the operators in Table I with the following complexity constraints:

- Predicates can take one or two time arguments and any number of object arguments.
- Only the operators \( \Box(t_1, t_2), \Diamond(t_1, t_2), \bullet, \lor, \land, \neg \) and \( \mathcal{K}_A \) are used.
- \( \mathcal{K}_A \) operators are not nested more than twice.

The queries on the future, which can be checked by the robot, will be constrained as follows.

**Definition 3.** Let iTED\(_f\) be defined as a subset of future tense logic formulae using the operators in Table I with the following complexity constraints:

- Predicates can take one or two time arguments and any number of object arguments.
- Only the operators \( \Box(t_1, t_2), \Diamond(t_1, t_2), \bullet, \lor, \land, \neg \) and \( \mathcal{K}_A \) are used.
- \( \mathcal{K}_A \) operators are not nested more than twice.

iTED\(_f\), jointly with iTED\(_p\), can express many useful performance requirements on a robot’s operation. Decidability on the satisfaction of these, in view of some autonomous decision by the robot, is a fundamental need of robot development. The following result states the conditions under which such decidability is achievable.

**Theorem 1.** Let \( L \) be a finite set of formulae in iTED\(_p\)∪ iTED\(_f\) with a limited future time horizon \( T \). Then \( L \) is strongly decidable from statements in \( S_t, F_{i,T} \).

**Proof.** The complexity of \( \eta \in L \) varies from single predicates to expressions involving the temporal, logical an epistemic operators. The proof is based on first checking that single current predicates are strongly decidable, which is the case as \( S_t, F_{i,T} \) represents the complete history of predicates up to time horizon \( T \). The rest of the proof follows by mathematical induction. If \( \nu, \mu \in \text{TED}_1 \cup \text{TED}_2 \) then we can prove that \( \nu \land \mu, \nu \lor \mu, \neg \nu \), etc. are all decidable. Assuming that some propositional formula \( \alpha \) is decidable, then any of \( \Box(t_0), \Box(t_1, t_2), \Diamond(t_1, t_2), \bullet, \lor, \land, \neg, \Box(t_1, t_2) \), \( \Diamond(t_1, t_2) \), \( \bullet \) are also decidable. Finally, it can also be easily verified that logical connectives of the latter temporal expressions are also decidable as they are dependent on deciding there the validity of their variables, hence completing the proof.

Note that decidability of a specification requirement does not necessarily mean that they will always be satisfied. Satisfiability of a requirement formula \( p \in \text{TED}_1 \cup \text{TED}_2 \) depends on two components:

C: The robot’s capacity and ability to come up with a plan \( \Pi(t, T) \) and its consequences \( F_{i,T} \), which satisfy \( p \).

E: The environmental conditions that support or prevent the robot to be able to have a plan \( \Pi(t, T) \), which satisfy \( p \).

Environmental conditions may prevent to satisfy a requirement. The legal requirements of robot operations must provide for the responsibilities of the robot’s operator, in order to oblige them not to use the robot under adverse environmental conditions where its performance may be compromised.

III. Degradations

The set of all performance propositions will be denoted by \( \mathcal{P} \), which are formulae from iTED\(_f\)∪ iTED\(_p\), as defined in the previous section. The set of robot specifications for a robot \( R \) is a finite subset \( S[R] \subset \mathcal{P} \), in which each specification is some modal/temporal logic formula to state a property of \( R \). The closure of \( S[R] \) of with respect to the propositional logic connectives \( \land \) (and), \( \lor \) (or) and \( \neg \) (not) will be denoted by \( C_1(S[R]) \) that provide a rich language to express the properties
of a robot, not all of which may be satisfied by a robot due to its possible degradations and environmental circumstances.

As the properties and abilities of the robot may change in time, and in practice they will, there are a number of possible consistent Boolean evaluations of $C_I[S[R]]$, the set of these is denoted by $W_S = \{ b \mid b : C_I[S[R]] \rightarrow \{0, 1\} \}$, also called the possible worlds of robot capabilities. We can think of possible worlds as possible changes onboard the robot and in the environment that it operates in.

**Definition 4.** For any $a, b \in C_I[S[R]]$, the notation $a \models_w b$ if $a \rightarrow b$ holds for the evaluation $w \in W_S$.

The following assumption is fundamental and in practice easily satisfied.

**Assumption 1.** For any $a, b \in C_I[S[R]]$, $a \models_w b$ iff $a \models_w b$ for all $v, w \in W_S$, i.e. evaluations of implications of performance specifications are consistent across all possible worlds.

Due to this consistency assumption, we can drop the index from $a \models_w b$ and simply write $a \models b$ for any $a, b \in C_I[S[R]]$. This does not, however, mean that the evaluations themselves are uniform across all worlds, i.e. in view of changing onboard conditions and environmental circumstances they are not uniform. We only require that implications of specifications are consistent across all worlds, which is easily satisfiable in practice.

**Definition 5.** For any $s \in S[R]$ the set of degradations is defined by $D(s) = \{ q \in C_I[S[R]] \mid s \models q \}$.

This definition means that higher levels of performance specification imply weaker, degraded ones. Note that as logical consequences imply semantic ones but not necessarily vice versa as the logic we use does not model intrinsic physical properties of the world, which may also cause consequences beyond what logic can imply, i.e. $s \models r$ implies $s \models r$ but $s \models r$ does not always imply $s \rightarrow r$. For clarity we introduce the notation $L(s) = \{ q \in C_I[S[R]] \mid s \models q \} \subseteq D(s)$.

The following results shed light on some properties of degradations.

**Lemma 1.** For any $s, r \in C_I[S[R]]$ the following hold:

1. $D(p \land q) \subseteq D(p) \cap D(q)$
2. $L(p \land q) = L(p) \cap L(q)$
3. $D(p \lor q) \subseteq D(p) \cup D(q)$
4. $L(p \lor q) = L(\neg p \land q) \cup L(\neg q \land p) \cup L(p \land q)$

**Proof.** Note that, if both $p \models r$ and $q \models r$ hold, then $p \land q \models r$ and similarly, if $p \rightarrow r$ and $q \rightarrow r$ hold, then $p \land q \rightarrow r$, which imply 1). On the other hand, if at least one of $p \models r$ and $q \models r$ hold, then $p \lor q \models r$, and similarly, if at least one of $p \rightarrow r$ and $q \rightarrow r$ hold, then $p \lor q \rightarrow r$, hence proving 3).

2) follows from that for any proposition $r$, $(p \land q) \rightarrow r \equiv (p \land q) \land \neg r \equiv (p \land \neg r) \land (q \land \neg r)$, which means that $r$ is in $L(p \land q)$ exactly when it is in both $L(p)$ and $L(q)$.

Similarly, 4) follows from that for any proposition $r$, $(p \lor q) \rightarrow r \equiv ((p \land \neg q) \lor \neg r) \lor ((q \land \neg p) \lor \neg r) \lor ((p \land q) \lor \neg r)$, which means that $r$ is in $L(p \lor q)$ exactly when it is in at least one of $L(p)$ or $L(q)$.

An important consequence of the above lemma is that it clarifies that there can be other reasons for a degradation from $p \land q$ than those logically implied from both $p$ and $q$, i.e. it is possible for a specification $r$, that $p \land q \models r$ but not necessarily true that $p \models r$ and $q \models r$. Similarly, $p \land q \models r$ does not necessarily mean that $p \models r$ or $q \models r$.

Next we aim to investigate the structure of the set of all degradations. Based only on the above said, this problem becomes a pure evaluation problem over propositional logic, which is too general for our purposes. As we aim to grasp degradations of robot specifications, first we introduce physical dependencies over $S[R]$ and then solve the problem of characterising the set of all possible degradations in terms of tree graphs.

**IV. Dependencies of specifications**

Let $A, B, C \in S[R]$ be properties of robot $R$, which it needs to satisfy. It is often the case that there is dependency between “$A$ is needed” then “$B$ is needed” and “$C$ is needed”. For instance $A \models B$ would imply that if $B$ is needed to be satisfied when $A$ is needed as there is no physical world model, in which $A$ would be satisfied and $B$ would not be satisfied, hence the satisfaction of $A$ requires that of $B$. For instance, a certain range of an electrically powered drone needs a certain level of battery capacity.

**Definition 6.** A physical dependency relation $\models$ over a requirement set $S[R]$ for robot $R$ is a semantic implication, meaning that $A \models B$ iff in all possible worlds $A \rightarrow B, A, B \in S[R]$. If both $A \models B$ and $B \models A$ hold, then the notation $B \equiv A$ is used.

We assume that a number of dependencies $\models_d$ are known over $S[R]$ from understanding the physical models of the robot. Note that $\models$ is a transitive relationship by physics. If $\models_d$ is an initially known set of dependencies, then its transitive completion is $\models_d$, $\models_d$ may still not include all actual semantic implications $\models$ over $S[R]$. A complete physics based evaluation of $\models_d$ can be used to find all possible worlds relevant to performance, i.e. all possible evaluations of propositions in $S[R]$ and ultimately in $C_I[S[R]]$. If the $\models_d$ is incomplete then the set of possible worlds will be bigger than it needs to be and any requirement analysis of the robot will be more conservative as it will take into account a higher number of possible worlds. Here we will characterise all possible worlds as derived from known dependencies.

**Definition 7.** For any $S = S[R]$ and $\models_d$ let $G(\models_d) = (S, E)$ be a directed graph such that in its transitive completion $G = (S, E^*)$ for any of its vertices $v, w \in S$ there is an $e \in E^*$ connecting them iff $v \models_d w$ holds. The $G = (S, E)$ is called the minimal dependency graph over $S$.

A minimal dependency graph of a dependency relation may contain loops, i.e. a sequence of directed edges forming a closed path in the graph. On the other hand, if mutual physical dependency is discovered between the members of a subset of specifications, then they can be treated as semantically
equivalent performance statements, i.e. they imply each other in all possible worlds. For the analysis of robot performance, such equivalent sets should be combined to form a single statement and single vertex in the dependency graph.

**Definition 8.** Let \( G(\|=d) = [S, E] \) be a minimal dependency graph and \( S_x \subset S \) a maximal equivalent set in the sense that any edge connecting any two vertices \( v, w \in S_x \) is contained in the transitive completion \( E \) and there is no vertex in \( S \) outside \( S_x \) which could be added to \( S_x \) to preserve this property. Let define a new graph by merging all vertices in \( S_x \) into a single vertex \( \mu[S_x] \), while keeping edges going out from \( S_x \) or arriving into \( S_x \) for the single vertex \( \mu[S_x] \) formed. The new graph is called a semantic equivalence reduced graph, obtained from \( G \) by merging the vertices in \( S_x \).

Note that the above reduction step can be repeatedly applied until there are any semantically equivalent subsets in \( S \). In finite number of reduction steps one can arrive to a graph, which does not contain semantically equivalent vertex subsets with two or more members. The next lemma states the result.

**Lemma 2.** For any \( S = S[R] \) and \( \|=d \) the minimal dependency graph \( G(\|=d) = [S, E] \) can be reduced in a number of equivalence reduction steps to consists of a set of directed trees.

**Proof.** If there is no equivalence class remaining in \( S \) after a number of reductions steps, then the resulting graph \( S, E \) cannot contain a closed loop as otherwise the transitive extension of the loop would define an equivalence class with greater than one cardinality. On the other hand, any directed finite graph without loops consists of a set of directed trees or isolated vertices, hence proving the lemma. □

The resulting graph is called the *minimal tree graph* of \( S[R] \). \( \|=d \), denoted by \( M(S, \|=d) \).

**Lemma 3.** Any possible world \( w \in W \) can be obtained by an evaluation of the vertices \( M(S, \|=d) \) with Boolean values 0, 1 so that if \( v \in S \) is evaluated to 1 and \( w \in S \) to 0 then there is no edge connecting \( v \) to \( w \) in \( M \) which we called a plausible evaluation.

**Proof.** If \( v \models w \) then whenever \( v \) is evaluated to 1 then \( w \) is evaluated to 1 too, by definition of \( \models \). The reverse, that any plausible evaluation corresponds to a possible world, is not necessarily true as there can be physical arguments that \( E(u) = 0, E(v) = 1, E(w) = 0 \) is not possible and yet \( u \models v \) and \( u \models w \). □

**Corollary 1.** The \( M(S, \|=d) \) is a partially ordered set without equalities, the \( S \) itself is not necessarily so, it may contain equalities. All degradation sets \( D(s) \) can be reduced to directed trees within \( M(S, \|=d) \).

**Definition 9.** For any \( s \in S \) in \( M(S, \|=d) \) the set of vertices \( D^-(s) = \{ w \mid s \models w \} \) is called the minimal degradation set of \( s \).

If specification \( s \) cannot be maintained by the robot, then the best it could do is to regress to one of the specifications in \( D^-(s) \), or to adapt/reconfigure itself to one of these to be satisfied.

**V. Reconfiguration**

As modern systems become more complex, the control systems that regulate their performance must become increasingly complex. We demand high-performance, high-efficiency and high-tolerance to any failures. Ultimately this places a considerable demand on the engineers and system designers. Ideally these control systems should be able to adjust to any situation that they find themselves in, reconfiguring to “accommodate component failures automatically” [15].

Looze et al [16] states four key features that an automatic redesign procedure must address:

- “The failures are unanticipated”.
- “The available response time is limited”.
- “Non-standard control effectors and configurations may be required”.
- “The handling/ride qualities of the reconfigured [system] may be degraded”.

**A. Plug and Play**

A typical control system will have been designed for the operation of one specific plant. If the characteristics of the plant change at any time, the controller may well cease to operate satisfactorily [17]. Additionally new sensors or actuators may become available which make the current control algorithms sub-optimal, traditionally this would require a major redesign in order to incorporate the new components. Plug and play control [17], [18], [19] focuses on this area of new resources, rather than just dealing with component failure or adaptive control. Plug and play control focuses on two distinct challenges [17], first when a known addition is made and the control performance must satisfy prescribed requirements with a known control scheme. Secondly, when an unknown component is plugged in and a pre-existing controller design is not available. In this case awareness of the design is required in order to identify where the new component should be slotted into the control structure, for example using Youla-Kucera-based controlled modification [19].

**B. Reconfigurable and Polymorphic Control**

Polymorphic control [20], [21], [22] builds on distributive plug-and-play control. The prime aim of polymorphic control systems is to enhance key system properties, for example performance, resilience and fault-tolerance. This is achieved by modelling the underlying control system as a graph-like structure that is amenable to automatic analysis. The control structure is then part of the dynamic network, as it is fundamentally distributed amongst the sensors and actuators of the system. Therefore the controller is in a well-defined position, where it can be quickly reconfigured or restructured in both form and function.

Typically the polymorphic controller is based on a traditional block diagram, for example as built in Simulink, modelled as a graph. Individual components of the system are...
Any distributed computing framework provides a method for a collection of different program threads to exchange information across different machines connected within a network. For this reason it is essential in modern robotics where small, onboard embedded processing boards offer direct control and interface to actuators and sensors to provide system level capabilities. Additionally robots may need to communicate throughout the collective in order to achieve solve problems or achieve goals. Finally more power computing resources could be used in the wider network to provide extra processing capability.

Any distributed system is founded on the network on which it runs. An example of a distributed computing solution for robotics is the Robot Operating System (ROS). ROS is not a traditional Operating System. Rather it provides a structured communications layer in which individual processes can interact [11]. It simplifies the task of programming robots by providing a robust framework where the designer is provided a declarative programming environment for parallel computational processes of a robot. A ROS implementation of a robotic software has four typical components:

- **Nodes** - Nodes are basic processes that perform the sensing, computation and control tasks. Typically each node can contain several computational threads, although it may have additional sub-threads which the programmer is responsible for designing. Typical systems are formed from many nodes, each of which does a portion of the overall task.

- **Services** - Services provide a strict communication model where there is an established request and response message between two nodes. In a process similar to web services, a node may subscribe and subsequently request information via a service and then be supplied back with the information on demand.

- **Topics** - In order to publish messages any node can establish a topic and publish messages to it, as and when necessary. Any other node within the network may also publish to this topic. In order to receive messages, the other nodes may subscribe, wherein they can receive any message sent via a callback. A topic is a broadcast messaging stream and so does not provide any synchronous message transfer.

A fundamental difference between services and topics is that services are requester/provider initiated while topics are sender/provider initiated and the receivers are **immediately notified, asynchronously**. Both are however many-to-many communications as there can be several providers and receivers of any service or topic. Topics are inefficient when a node only needs some data from another node occasionally, when it needs it; while services are inefficient when a node needs some data supplied on a continuous, "as soon as possible" basis, though asynchronously. In their own way both are efficient ways to communicate for different purposes. Care needs to be taken however that a subscriber to a topic does not receive more data than it needs as otherwise it is wasting its computational resources on handling redundant messages from the topic. For instance sensor messages are to be published to a topic only with a frequency which is needed by other nodes, thereby resulting in less latency than if a service were doing the same job.

### A. Mathematical model of a ROS Package

There are two possibilities for describing a ROS based system:

- A tri-partite graph with vertices for nodes, topics and services.
- Nodes with labelled, directed edges for topics and services.

The second representation is less amenable to exact mathematical analysis. Hence our choice remains the first option, the tri-partite graph, where each of the three vertex types are not interchangeable in graph matching algorithms. New topics and services can be easily introduced that can allow reconfiguration of the system to provide agents with the information they
required, albeit sourced from different locations. This can be seen in Figure 1, where vertices represent nodes, topics and services. Edges show the routes of information flow, all services and topics therefore must have at least one incoming edge from a node. Additionally all node communication must occur through topics or services.

Definition 10. A ROS-graph is $G = \{N, T, S, E, D, C, X, \lambda\}$, where $N$ are the set of vertices representing ROS nodes, $T$ are a set of topics and $S$ are a set of services, $C$ is a partially order set of object classes and $X$ is a set of labels to name all vertices. $E \subseteq (N \times T) \cup (T \times N) \cup (N \times S) \cup (S \times N)$ is a set of directed edges to represent publishing of, and subscription to, topics and provision of, and subscription to, services, respectively. $D : E \rightarrow C^\ast$, $E^\ast = T \cup (N \times S) \cup (S \times N)$, is a data descriptor function where $C^\ast$ is a notation for finite sequences of entries from the set of data object classes $C$, which are used in services and topics to send information between nodes. Each of $N, T, S$ are labelled by a surjective labelling function $\lambda : N \cup T \cup S \rightarrow X$.

Hence a ROS system focuses on the ability of nodes to advertise or utilise services, and to publish or subscribe to topics. $G$ represents the maximum ability of the robot when the system has all nodes, topics and services nominally functioning. If some nodes are not available due to sensor, actuator or computational hardware breakdown, then $G$ needs sufficient redundancy to enable continued functioning of the robot or at least some of its functionality. The ROS graph $G$ defines all the possible data flows for sensor readings, signal processing and control action in the environment. A detailed description is not within the scope of this paper and we refer the reader to [10].

From the point of view of system reconfiguration, we are interested in time dependent variations of a nominal ROS graph: the active ROS sub-graph may change in accordance with the availability of sensor, actuator and computational nodes in the processor network of the robot on a time-line $t \geq 0$.

Definition 11. A time dependent ROS sub-graph $G(\alpha_t)$ of a nominal graph $G = \{N, T, S, E, D, C, X, \lambda\}$ is defined with the property that at any time $t \geq 0$ the equivalence $\{(v, n) \in E(\alpha_t) \} \leftrightarrow \{(v, n) \in E \& \{\exists \mu \in N(\alpha_t) : (m, v) \in E(\alpha_t)\}, \forall v \in S(\alpha_t) \cup T(\alpha_t), is satisfied and in general $N(\alpha_t) \subseteq N, T(\alpha_t) \subseteq T, S(\alpha_t) \subseteq S, E(\alpha_t) \subseteq E, D(\alpha_t) \subseteq D, C(\alpha_t) \equiv C, X(\alpha_t) \subseteq X, \lambda(\alpha_t) \subseteq \lambda$ hold.

This definition simply requires the consistency condition between service/topic provision and availability.

B. Necessary concepts in reconfiguration

Given a time varying ROS-graph, $G(\alpha_t)$, the most basic requirement of reconfiguration is that the reconfigured system should remain to fulfill a prescribed set of functionality. A reconfiguration algorithm of a robotic agent is concerned with amending the sensing/action/planning skills of a robot in such a manner that its goal achievement capability is at least partially maintained. For our formal analysis we assume that the robot can achieve a set of high level goals supported by sets of sub-goals in layers which are ultimately supported by physical and perception/modelling/planning skills in terms of services/topics. At the lower end of this dependency are services or topics for sensing and actuation using hardware devices, various computations in the middle and skills higher. To formalise this dependency of goal achievement and skill-based ability on lower level processes, we introduce the following definition.

Definition 12. Let $F = F(G)$ be a set of algorithms with i/o object classes in the C of a ROS graph $G = \{N, T, S, E, D, C, X, \lambda\}$. A functional dependency graph $F$ is a partially ordered set $F = \{Q, \prec\}$ where $a \prec b$ means the b depends on a algorithmically as it is used by b. A functional assignment $\Psi : Q \rightarrow 2^S \cup 2^E$ is a specification of what service or topic can provide a functionality in $Q$. If $\Psi(q), q \in Q$, is a service set then we also define $\overline{v}(q) \in C^\ast \times C^\ast$ and if $\Psi(q), q \in Q$, is a topic set then $\overline{(q)} \in C^\ast$ for the i/o data of algorithms in $Q$.

The set of permitted assignments also depends on the details of algorithms and not only i/o compatibility of service and not only on object classes in topic broadcasts. This is grasped by the set of all acceptable realisations which depends on knowledge of the functional assignments $\Psi$, as time progresses, as follows.

Definition 13. A functional realisation $\mathcal{R} : Q \rightarrow E(\alpha_t)$ is a surjective map where $E(\alpha_t)$ is defined by $E(\alpha_t) = \{e \in E(\alpha_t) : e = (n,v), n \in N, v \in T \cup S\}$ from $G$ and $\Psi \in Q : \mathcal{R}(q) = (n,v) \rightarrow v \in \Psi(q)$ is satisfied. The set of all acceptable realisations of $F$, based on $G(\alpha_t), \Psi$, is defined by $\mathcal{R}_F(\Psi, G) = \{\mathcal{R} \mid \mathcal{R} : Q \rightarrow E(\alpha_t) is a functional realisation\}$

Functional realisation essentially defines the relationship between the functionality represented by the set of algorithms $Q$ and computational hardware. Several service realisations (potentially in different nodes) can represent equivalent or similar functionality for $Q$ to provide redundancy of hardware and computational resources. Not all services and topics directly handle a device. Services can exist between goals and devices in the functional dependency graph for modelling the environment, for planning and reasoning to comply with

![Fig. 1: Basic ROS-Graph. Unidirectional edges can denote publication of or subscription to topics or provision of, and request of, services.](Image)
behaviour constraints. Members of $Q$ can be device drivers, sensor signal processing, modules for simultaneous localisation and mapping (SLAM), physical skills of mobility or planning and also sub-goals such as handling objects, tracking a planned path of movements.

The $F(G)$ (henceforth called FD-graph for short) is however more than $Q$, it also defines the dependency relation $<$ of algorithms, not yet considered in terms of realisations defined so far. The equivalence of $a \in Q$ with $b \in Q$ for any other algorithm, which can use them, is a fundamental and simple redundancy relationship on which reconfiguration theory can be built.

**Definition 14.** Let $F_G = (Q, <)$ be a functional dependency graph. A functional equivalence relation $R$ is a subset $R \subset \{(a, b) : \exists c : a < c, b < c, a, b, c \in Q\}$ with the properties of reflexivity, symmetry and transitivity and $(a, b) \in R$ is denoted by $a \approx b$.

$a \approx b$ is used to express that $a$ and $b$ serve the same purpose but with possibly different algorithmic implementations. If $a \in Q$, the equivalence set of $a$ is defined by $\sigma(a) = \{b \in Q : a \approx b\}$. The concept of functional equivalence gives the opportunity for a reconfigurable system to use two or more different algorithms with the same input and output formats but possibly different performance characteristics. For instance nonlinear optimisation can be implemented by a number of methods but they can be initiated with the same format of input and return data of the same format. Similarly, automated robust controller design, various dynamical modelling techniques, planning and SLAM algorithms can have the same object classes for inputs/outputs and can be functionally equivalent.

Due to transitivity, equivalence classes are formed within $Q$. Each of the elements in an equivalence class can be represented by multiple services in various nodes to provide redundancy for the possibility of reconfiguration in case of some malfunction occurs in the hardware/software modules. The process of reconfiguration is simplified due to a property of “independence” ensured by the above definitions in the sense that service/topic provision can replace each other if they serve equivalent functionality on $F(G)$.

**Lemma 4.** A ROS graphs $G$ and an associated $F(G)$ are independent in the sense that if $a \approx b$, $(a \rightarrow (n, v)) \in R$ and $(b \rightarrow (m, w)) \in \bar{R}$ in an acceptable realisation $\bar{R} \in \bar{R}(\Psi, G)$ then there is some acceptable realisation $\bar{R}' \in \bar{R}(\Psi, G)$ so that $(a \rightarrow (m, w)) \in \bar{R}'$ and $(b \rightarrow (n, v)) \in \bar{R}'$.

Proof. By definition the $\bar{R}(\Psi, G)$ we generally have $\forall q \in Q : \bar{R}(q) = (n, v) \rightarrow v \in \Psi(q)$ which implies $\bar{R}(a) = (n, v) \rightarrow v \in \Psi(a)$ and $\bar{R}(b) = (m, w) \rightarrow w \in \Psi(b)$, in particular. As $\Psi(b) = \Psi(a)$ due to $a \approx b$, this means that a realisation, in which $(a \rightarrow (m, w))$ and $(b \rightarrow (n, v))$, is also acceptable. □

The equivalence relation $a \approx b$ on an FD-graph provides it a rich structure for reconfigurability under ROS, which the next subsection will analyse.

### C. Use of functional dependency

Computation for reasoning and high level planning can be outside the set of services and topics and can be located in a “leading” node under ROS for a robotic agent. In such an implementation, this reasoning node of the robotic agent can handle the set of goals and subgoals, sets priorities by using goal lists, modelling and planning services. In our current ROS model such a reasoning ROS-node would not provide any services. Let denote the functionality of reasoning by $r \in Q$. Reasoners could also be represented by “equivalent” vertices in the functional dependency graph $F = (Q, <)$ but without providing dependency to any other functionality. If $r \in Q$ is a reasoning functionality in $F$, then there is no $q \in Q$ such that $r < q$. More importantly, $r$ should be possible to realise using more than one ROS-node, thereby providing redundancy. To enable the formalism of realisation introduced above, we add the “idle” service $\emptyset$ to $S$ so that henceforth $\emptyset \in S$ will be assumed. This enables various reisations $r \rightarrow (n_1, \emptyset)$, $r \rightarrow (n_2, \emptyset)$ of a reasoner $r \in Q$. Reasoning is at the top of the relation $<$ for dependency on $F(G)$. The “nearly maximal” vertices in $M_F = \{s \in Q : r > s \}$ can be high level skills to achieve some goals of the robotic agent. At the lower extreme of $(G(F)$, minimal vertices $m_F = \{s \in Q : q > s\}$ of the dependency graph $F$ are services or topics for device operations such as sensors and actuators.

The functionality of the reasoner $r$ and other algorithms in $F(Q)$ are mapped onto a ROS graph for implementation using a realisation. As each functionality in $Q$ can be possibly represented by one or more resources in the associated ROS system, there is redundancy and reconfiguration is facilitated. Hence redundancy can be twofold: the equivalence of functionalities can create alternatives within $Q$ as well as the implementation in ROS can vary for each of the functionalities in $Q$.

### VII. Structure of FD-Graphs

Let $F$ be an FD-graph with vertex set $Q$ and directed edges represented by a partial order $<$ and associated with a ROS graph $G$ through some functional assignment $\Psi$ so that $F = F(G)$. For any $c \in Q$ the set of algorithms, which $c$ may need, also called the functional support of $c$, will be denoted by $\Delta(c) = \{a \in Q : a < c\}$. $\Delta(c)$ is the maximal set $\Delta(c) \subset Q$ such that $c \in \Delta(c)$ and $\forall a, b \in Q : a < c \& b < a \rightarrow b \in \Delta(c)$. For any $c \in Q$ the set of equivalent classes over $(Q, \approx)$, from which at least one algorithm is used by $c$, is denoted by $\eta(c)$. A $\delta(c) \subset Q$ is called a functional basis of $c$ if it contains exactly one element from each entry in $\eta(c)$. There can be several or even a large number of functional basis for each $c$, depending on the richness of the equivalence relation $(Q, \approx)$.

The essence of reconfiguration of a robot under ROS is to enable a capability $c \in Q$ to find a ROS-realisable functional basis $\delta(c)$. Breakdown in the availability of hardware can be caused by some sensing, actuation or computational node failing. These can happen either due to a software breakdown of a node, which can happen independently from other nodes, or due to sensor/actuator hardware failure. The time dependent ROS-subgraph $G(\alpha_t)$, as defined in Definition 11, will enable or deny the existence of a functional basis $\delta(c)$. If
\(G(\alpha_t)\) supports the existence of a functional basis \(\delta(c)\) for a \(c \in Q\) then the \(c\) is called configurable under \(G(\alpha_t)\). If all \(c \in Q\) are configurable then the functionality of \(F(G)\) is called fully configurable in \(G(\alpha_t)\). The problem is when \(G(\alpha_t)\) changes and reconfiguration is needed in terms of a realisation \(\mathcal{R} \in \tilde{\mathcal{R}}_n(\Psi, G)\) as in Definition 13. The main issue is the computational complexity of finding a full relisation for given \(G(\alpha_t)\) and \(Q, \approx, \Psi\) : is it better to centralise this, for instance through the reasoner \(r?\) - or is it better to create a distributed system where functionalities and nodes take care of themselves?

A. Realization of partial capabilities

Finding a functional basis for a \(c \in Q\), which is realisable in \(G(\alpha_t)\), is a combinatorial problem. One way to reduce its complexity is going backwards: reduce first the equivalence classes in \((Q, \approx)\) to those which are supported by \(G(\alpha_t)\) and then "intersect" that with \(\eta(c)\) (the equivalent classes supporting \(c\)) to get a realisation of \(c \in Q\). The complexity problem is that reducing the equivalence classes in \((Q, \approx)\) to those which are supported by \(G(\alpha_t)\) means a realisability check for each of \(a < c\). It can also happen that some \(a \in Q\) is not found realisable in \(G(\alpha_t)\). For instance, due to hardware failure, the robotic agent will be able to achieve only parts of the set of goals it has been programmed to achieve.

In practice the mission of a robot should not be a total failure because of partial capability loss. The framework of reconfigurability does, however, need to provide the ability to the robot to reconfigure if that is possible, meaning the ability of finding a realisation \(\mathcal{R} \in \tilde{\mathcal{R}}_n(\Psi, G)\) for any \(c \in Q\) which could in principle be enabled.

B. Complexity of centralised reconfiguration

The notation \(\kappa(S)\) will be used for the cardinality of any set \(S\). In a centralised approach to configuration, where only the time varying ROS graph \(G(\alpha_t)\) is available, the number of algorithmic steps of configuration can be analysed. The computational effort, in terms of number of operations, for checking the configurability of \(c \in Q\) will be denoted by \(\Xi(c), c \in Q\) for given \(G(\alpha_t), \Psi, Q, \approx\).

**Lemma 5.** For any \(c \in Q\) the worst-case computational complexity configurability can be recursively computed as

\[
\Xi(c) = \Pi_{S \in \Psi(c)} \sum_{\alpha \in S} \Xi(\alpha) \quad (1)
\]

if \(\eta(c) \neq \emptyset\) and \(\Xi(c) = \kappa(N(\alpha_t)) \times \kappa(\Psi(c)) \times \log(\kappa(E(\alpha_t)))\) is defined for any \(c \in Q\) such that \(\eta(c) = \emptyset\), i.e. there is no \(a \in Q\) with \(a < c\).

**Proof.** For \(\eta(c) = \emptyset\) there is a simple test to be carried out whether \(\exists n \in N(\alpha_t), \exists v \in \Psi(c) : (n, v) \in E(\alpha_t)\) which amounts to \(\kappa(N(\alpha_t)) \times \kappa(\Psi(c))\) operations of membership testing, most efficiently done by binary search. For \(\eta(c) \neq \emptyset\) one needs to examine whether any of the equivalent algorithms needed are configurable, which leads to (1).

Assuming that the \(\Gamma = \max_{c \in Q} \kappa(\sigma(c)), \bar{\Gamma} = \max_{c \in Q} \kappa(\eta(c))\), \(\tilde{\Phi} = \max_{c \in Q} \kappa(\Psi(c))\) and \(M = M(\Theta, <)\) the length of the longest path in the FD graph, the following complexity result holds for centralised decisions on reconfigurability.

**Theorem 2.** The worst-case complexity of a top-down centralised testing procedure for reconfigurability is bounded by

\[
\Xi(F, G, \Psi | G(\alpha_t)) \leq \kappa(Q) \times (\Gamma \times \bar{\Gamma})^M \times \kappa(N) \bar{\Phi} \times \log(\kappa(E)) \quad (2)
\]

**Proof.** Repeated application of (1) and the definition of \(\eta(c)\) for \(\eta(c) = \emptyset\) results (2) by mathematical induction for \(N = 1, 2, 3, \ldots\)

C. Decentralised organisation of reconfiguration

The above top-down centralised operation of (re)configuration of ROS resources for functionalities is computationally intensive in case of failures. This section outlines a bottom up, distributed computational approach to provide parallel monitoring, for each \(c \in Q\) the availability of some computational node in the ROS system \(G(\alpha_t)\). If \(\Psi(c) \cap (S(\alpha_t) \cup T(\alpha_t)) = \emptyset\) then monitoring of configurability is irrelevant as there are no resources to support \(c\). It is therefore only worthwhile monitoring options of live capabilities and the failure of a capability so that the agent reasoner is well aware to influence its decisions. This means that each service and topic on each node needs to monitor all the service and topic resources, which it may need to rely on, for their availability. This is a considerable computational overhead as it requires for any \(c \in Q\) carried out by for instance by \((n, v), n \in N, v \in T \cup S\), that \(n\) monitors by Booleans the availability of any algorithm from \(\eta(c)\). This requires \(n\) to keep a Boolean register of nodes across the ROS network which can provide the functionality for any capability in the equivalence classes of \(\eta(c)\).

**Lemma 6.** A parallel system of reconfiguration requires, in the worst case, each node \(n\) with a total number of \(\kappa = \kappa(S(\alpha_t) \cup T(\alpha_t) \cup (n, v))\) to maintain \(\kappa \times \tilde{\Gamma} \times \bar{\Phi}\) number of Boolean registers for the availability, selection and replacement of any functionality needed by \(n\) in some of its services or topics.

**Proof.** In the worst-case the number of different replacements, which may be needed by any service/topic, is \(\tilde{\Gamma} \times \bar{\Phi}\), this multiplied by the number of services/topics int he node gives the result.

This functionality can be ensured by periodic broadcasts (topics) by each computational unit about their availability to all nodes which may needed them in the future. If they are not available then they do not broadcast and hence all nodes are informed about the availability of resources they depend on. Even more importantly, if a service or topic falls out, they may be able to replace them by a functionally equivalent one. The amount of communication to inform availability across the ROS network is high, the advantages and disadvantages are discussed in the next subsection.

D. Centralised versus distributed reconfiguration

The pros and cons between a centralised assessment of reconfiguration versus decentralised one is as follows:
1) **Communications overheads.** In both the centralised and decentralised approaches all computational units in need to broadcast their health but in a centralised approach a single reasoning-node will receive these messages while in a decentralised approach only those which functionally depend on them. On the other hand, the decentralised approach has the overhead of larger number of communications broadcasts carried out across the live ROS system periodically, loading the system with large amounts of communications perhaps never needed due to reliability of components.

2) **Speed of decision how to reconfigure.** Clearly $\mathbb{E}(F(G, \Psi | G(a)))$ can be a large number. Even if a single availability check takes a small fraction of a second, when $\varphi$ exceeds millions then real-time reconfiguration is unsustainable in a centralised manner. On the other hand, the decentralised approach has the availability data of computational resources it may need and can switch to the use of them in realtime.

3) **Robustness to hardware failures.** Avičienis et al [27] define robustness as ensuring dependability and all its facets including availability, safety and maintainability. A simpler definition is “the ability of the system to meet its requirements under a range of representative failure conditions” [28], [29]. In this case this is a useful definition as we require the autonomous robot to be able to reconfigure, given a failure condition and be able to lay down a new ROS-graph allowing for continued operation.

In the centralised approach the node which computes the reconfiguration of hardware resources presents a bottleneck. Any node failure will prevent the system from being able to reconfigure, as this node solely is responsible for providing this capability.

On the other hand, in a decentralised approach, whichever nodes are affected, parts of the system will survive and will be able to provide compromised capabilities to satisfy a mission. Under ROS communications of services and topics to, or from, a node would only suffer if a whole processor is goes down. To avoid the latter, again the number of processors available needs to be increased to facilitate a decentralised approach to ensure redundancy and remove this single point of failure.

The addition of extra processing components increases the communications overhead required under a decentralised approach. However, if this can be secured, the decentralised approach is superior to centralised decision making for self-reconfiguration.

**E. Optimal Redundancy in Large Systems**

Independence of $G$ to $F(G)$ in Lemma 4 means that ROS implementations of algorithms can be arbitrarily “plugged in” by matching nodes as services/topics to algorithms as long as the assignment $\Psi : Q \rightarrow 2^S \cup 2^T$ permits. System schemas can be constructed by the configuration agent to obtain a functioning ROS graph for implementation. The greater the reconfigurability in face of unavailable nodes the greater the power of this approach. Here we present a quantitative analysis in case a large number of components are needed to build up an autonomous system or robot.

Assume that the probability of the availability of a functionality $c \in Q$, which can rely on a piece of hardware, is denoted by $p(c)$. We are interested to obtain the probability of the robotic system not being disabled by lack of reconfigurability. To investigate the quantitative relationship between the probability of component failure, level of redundancy and overall likelihood of breakdown, in this section we assume that any equivalence class $\sigma(a)$, $a \in Q$ has $k(\sigma(a)) = p$ number of elements and each of these have $\mu/\rho$-fold redundancy than the number of variations for providing the functionality corresponding to $\sigma(a)$ is $\mu$.

For the sake of investigating the relationship between the number of components in $Q$ and the corresponding redundancy level to maintain a certain probability level of reconfigurability, we assume that an integer $\mu \geq 1$ now denotes our uniform multiple redundancy level.

**Theorem 3.** For a given probability $q \approx 1$ of reconfigurability we can state the following:

1. If $q$ is required and $N \rightarrow \infty$ then the redundancy level $\mu$ needed is proportional to $\log N$.
2. If $N$ is fixed and $q \rightarrow 1$ then the required redundancy is asymptotically proportional to $\log \log q^{-1}$.
3. Otherwise an approximate formula for the redundancy level to ensure reconfigurability with probability $q$ is

$$\mu = \frac{\log(- \log q)}{\log p} - \frac{\log N}{\log p}$$

**Proof.** The essence of the proof is based on discovering what happens to $r$ to maintain the relationship

$$(1 - p^r)^N = q$$

This leads to

$$\mu = \frac{\log \left(1 - q^{\log q/N} \right)}{\log p}$$

Given that $\log q/N \approx 0$ the approximation $\exp[\log q/N] \approx 1 + \log q/N$ can be made that results (3) as

$$\mu = \frac{\log(- \log q)}{\log p} - \frac{\log N}{\log p}$$

which also implies (1) and (2) when $N \rightarrow \infty$ or $q \rightarrow 1$. $\square$

If control system reconfiguration is needed due to changing resources (either during robot operations or at design stage) the process of finding available components for the system graph can be automated.

**VIII. Examples of Reconfiguration**

This section will consider the application of reconfiguration to an Unmanned Aircraft System (UAS). Reconfiguration in the operation of a UAS remains a critical component in enabling operation in different flight regimes, for example when switching from Visual Line of Sight (VLOS) to Beyond Visual Line of Sight (BVLOS) [30]. In the BVLOS flight regime the...
decisions are made onboard the aircraft, thus the situational awareness must be transferred from the pilot to aircraft. In the VLOS case the pilot augments the aircraft, under BVLOS the aircraft must provide all sensing requirements required for artificial situational awareness [31], [32], [33], however, onboard sensors can be combined to provide to quantify their effectiveness [34].

A. Scenario

A number of commercial drone operators in the UK regularly undertake structural inspections with their vehicles. When flying in close proximity to a structure it is not possible to rely on Global Navigation Satellite System (GNSS) for positioning due to its limited accuracy and susceptibility to occlusion and multipath errors [35]. Additionally, it is unlikely the vehicle will possess a high fidelity geofence [36] (corresponding to the physical outline of the structure) required for a GNSS system to assureadequate separation. Pilots must instead position the vehicle manually using an attitude stabilisation mode [37], requiring significant skill.

This piloting method introduces a significant delay between the vehicle experiencing a disturbance and corrective action being taken by the pilot. This delay imposes strict limits on the operating conditions for the vehicle so as to maintain safe separation from the structure at all times. The maximum wind gust which can be tolerated is primarily determined by the reaction time of the pilot which can be variable due to the high workload associated with the task. Additionally, the maximum distance from the pilot the vehicle can be operated is limited by the pilots ability to adequately judge the separation distance.

The semi-autonomous controller discussed in this section is tasked with significantly reducing pilot workload during an inspection task while maintaining a fixed distance and relative heading to the structure at all times. The controller utilises a LiDAR scanner to detect the structure and takes control of the vehicle around the pitch and yaw axes away from the pilot, Fig. 2.

![Fig. 2: Diagrammatic representation of the control task.](image)

System running the ArduCopter firmware\(^1\). The autonomous control is deployed to an additional onboard computer, a raspberry Pi3 running Linux and the Robot Operating System (ROS) [11]. An additional NVidia Jetson TK1 is also available to provide flexibility and availability of higher-performance computing where necessary. Fig 3 illustrates a schematic view of the onboard systems, with the autonomous controller executing on the onboard processor, a high-powered directional WiFi-bridge provides connectivity to the UAS whilst in flight.

Fig. 4a shows the components on the actual vehicle. We have used this platform previously to show how a model based design framework can produce verifiable software [38], which is demonstrated in Fig 4b. In this case successfully detecting errors in the implementation of an autonomous building inspection drone, enabling the software implementation to be verified against the task to be completed.

In Section II-A1 a number of examples of logic for TED were formed for common UAV applications. In this case we are interested in the the pair of statements below which capture an inspection task taking place:

$$\square:\forall t(\text{TakingOffAtLocation}(L_1) \land \text{Inspect}(O_n^{ENV}) \rightarrow \exists \phi(p^f(O_n^{ENV})))$$

Representing the existence of a plan, \(p^f\) to inspect object \(O_n^{ENV}\) whilst the aircraft is airborne.

$$\square:\forall t(\text{TakingOffAtLocation}(L_1) \land \forall O^{ENV}_{n}$$

$$\land \exists(d_{n}^{ENV} \neq d_{n}^{inspection}) \land p^f(O_n^{ENV}) \rightarrow$$

$$\exists \exists \text{maintainDistance}(O_n^{ENV}, d_{n}^{ENV_{inspection}}))$$

Symbolising that, whilst airborne, a UAV should not move closer or be further from an object, \(O_n^{ENV}\), than a set inspection distance, \(d_{n}^{ENV_{inspection}}\) when there exists a plan to inspect the object, \(p^f(O_n^{ENV})\). If this is the case a controller should be used to maintain the distance, \(\text{maintainDistance}(O_n^{ENV}, d_{n}^{ENV_{inspection}})\).

The LiDAR provides the distance of the UAV to the object being inspected, \(O_n^{ENV}\). The performance of the LiDAR has a direct dependency on the capability to maintain distance, \(\text{maintainDistance}(O_n^{ENV}, d_{n}^{ENV_{inspection}})\). The performance of a range of different Hokuyo LiDAR devices is shown in Table II. As can be seen from Table II there are a wide range of different performing devices that can be selected, additionally as devices degrade there are a wide range of criteria that may begin to affect performance.

Initially let us consider the Hokuyo URG-04LX-UG01. The performance of the sensor will define how well \(d_{n}^{ENV_{inspection}}\) can be maintained. The performance of the controller will be determined by accuracy, the angular resolution and the scan time of the device.

The controller can be selected to be as complex or simple as required. Typically a PID controller will suffice, this controller uses error relative to a set-point to modify vehicle control inputs to maintain position relative to the wall, typically this

\(^1\text{http://copter.ardupilot.com/wiki/common-pixhawk-overview/}\)
is done in two components distance and relative angle. The controller structure is a typical PID with a set of gains tuned to be appropriate for the underlying control systems. However, the set-points, especially $d_{ENV}^{inspection}$, can be selected to be appropriate for the device and task.

The TED logic specifies that the aircraft will perform the assessment and pass within a set distance of an object providing that there is a plan to inspect it. Logic formulae can now be written to incorporate any change in measurable sensor performance.

\[ \Box (t) \forall L_1 (TakingOffAtLocation(L_1) \land \neg \Box (t) ScanTime(t_{scan} < t_{min}) \rightarrow \exists \Box (t) ScanTime(t_{scan} < t_{min}) ) \]

This symbolises that if there is a plan to inspect an object the scan time provided by a sensor must fit a timing constraint where the scan time $t_{scan}$ is less than a threshold value, $t_{min}$. Then a monitor placed on the scan time can then be used to modify the set-point for the controller to a longer distance, $d_{far}$ if scan times are exceeded, or set to close inspection if scan times are appropriate:

\[ \Box (t) \forall L_1 (TakingOffAtLocation(L_1) \land \neg \Box (t) ScanTime(t_{scan} < t_{min}) \rightarrow \exists \Box (t) ScanTime(t_{scan} < t_{min}) ) \]

\[ \Diamond (t) \forall L_1 (TakingOffAtLocation(L_1) \land ScanTime(t_{scan} > t_{min}) \rightarrow \exists (t) ScanTime(t_{scan} < t_{min}) ) \]

IX. Conclusion

This paper lays the foundation for autonomous reconfiguration within ROS. This paper describes how control systems for robots can be realised within the Robot Operating System (ROS), providing options for reconfiguration. This is based around a graph representation of a ROS system which is marked up to include labels and descriptions that allow the components to be matched. This has been extended to include a time-dependent component which provides more flexibility into the system configuration; allowing configuration to naturally adjust through time or for changes in activity during operation.

It has generally been assumed that the assessment of performance of signal processing for perception and control is not part of the reconfiguration process: such an approach assumes that modules for perception and feedback control are pre-designed adaptive systems, which are self-initialising and are ready to be used after activation in realtime.

In an example we have formulated the graph based architecture within a reconfiguration problem within PDDL.
and also illustrated the use of the decentralised approach to reconfiguration.

The authors would like to thank...

REFERENCES


[34] ——, “Quantifying situation awareness for small unmanned aircraft towards routine beyond visual line of sight operations,” *The Aeronautical Journal*, in press.


**Sandor Veres** holds a chair in Autonomous Control Systems and leads the Robotics and Autonomous Systems Research Group at the Department of Automatic Control and Systems Engineering, University of Sheffield. Between 2002 and 2012 he held the chair of Control Engineering at the School of Engineering Sciences, University of Southampton, where his funded research interests were system identification for control, vibration control, satellite formation flying, formal verification of autonomous underwater vehicles and engineering of autonomous spacecraft software. His currently funded research projects are on reconfigurable autonomy, distributed sensing, control and decision making in multi-agent systems, verifiable autonomy, learning by autonomous cars, safety verification of unmanned micro-air vehicles and mission management of autonomous surface boats. He has published 4 books and about 240 refereed papers on a variety of topics in control sciences and intelligent systems.

**Jonathan M. Aitken** is a University Teacher in Robotics in the Department of Automatic Control and Systems Engineering at the University of Sheffield. He received the Ph.D. degree in Electronic Engineering from the University of York, United Kingdom in 2010. He is recognised for his work on autonomous robotic reconfiguration, analysis and engineering of distributed systems, robotic co-working and system identification for control. He has published over 30 peer-reviewed research publications across these areas, and actively

**Owen McAree** Biography text here.