A MULTI-AGENT APPROACH TO INTEGRATED FDI & RECONFIGURATION OF AUTONOMOUS SYSTEMS

Badril Abu Bakar and S.M. Veres
Faculty of Engineering, Science and Mathematics, School of Engineering Sciences
University of Southampton
SO17 1BJ
Southampton, U.K.
email: bhab1g08@soton.ac.uk

ABSTRACT

This work is concerned with a new type of realtime reconfigurable control systems that is based on the use of autonomous agents. To this end, two stages have been examined in the context of intelligent agent decision making; the fault detection and identification (FDI) stage and the reconfiguration stage (RC). The FDI stage detects that a fault has occurred. It then further diagnoses the situation. The RC stage follows this by adapting or changing the control architecture to accommodate the fault. The problem is to synchronize or integrate these two stages in the overall structure of a control system on real world applications. The multi-agent architecture proposed in this paper has several advantages in terms of modularity, reliability, ability to learn and achieve overall higher robustness over past "single software" methods. Initial tests on an example system have been carried out to demonstrate this new organisation of reconfigurable control systems.

KEY WORDS
Reconfigurable control, FDI, Intelligent systems, Machine learning, Control application, Multi agent

1 Introduction

Automated controllers provide the means for manipulating a plant's behaviour in a desired manner. A properly designed controller prevents the plant from operating in a dangerous and unstable mode due to malfunction of hardwired components. A poorly designed controller on the other hand, will result in significant damage to the plant and/or injury to people. Growing demand for safety, reliability and survivability in automated and autonomous systems have called for designing controllers that can absorb these imperfections in design and conditions. Conventional robust controller designs have become increasingly inadequate to cater for these complex systems [1].

The design of such controllers can be approached in many ways. Fault tolerant control systems (FTCS), as this field is termed, has been rich with research activity. In the literature, reconfigurable control systems fall under the scheme of active fault tolerant control systems (AFTCS) [2]. It is distinguished from its counterpart, passive fault tolerant control systems (PFTCS) through the integration of both the fault detection and identification scheme (FDI) and a reconfigurable controller (RC) in the overall structure [3]. Most of the research done in this field have been motivated by aircraft flight control designs. Adaptive control and robust control are just two of the many approaches that have been developed [4][5][20] [7][8][9][10][11][12]. Due to the complexity of the problem, research in the past either concentrated on the fault detection and identification part (FDI) or on the control reconfiguration part (RC). Only a few research papers have been devoted to both components in the same framework.

The problem our work investigates can be summarized as follows. An online reconfigurable controller is desired due to (1) malfunction of actuator components (2) partial malfunction of sensor components. To this end, two stages have to be looked at; the fault detection and identification (FDI) stage and the reconfiguration stage (RC). The FDI stage detects that a fault has occurred. It further diagnoses the situation. The RC stage on the other hand, adapts or changes the control architecture to accommodate the fault. The problem is to synchronize or integrate these two stages in the overall structure of a control system in real world applications.

Efforts to combine these two stages have been made in the past as described in section 2. However, an effective integration in practice still poses a challenging problem. These problems include deciding when is the right time to reconfigure; the accuracy of the FDI component in providing information to the RC component; modelling mismatch between the two components and critical timing issues. This is largely due to the fact that embedded subsystems usually work well alone, but face difficulties providing adequate and timely response to other components when working together.

Here a novel agent based approach is proposed. A multi agent architecture to solve the integration of FDI and RC will be introduced. This distribution of tasks allow agents to be more concentrated in their effort to achieve their goals. It allows developers to take a more modular approach towards designing a reconfigurable controller. Our approach incorporates machine learning algorithms for

some agents. This allows the overall system to learn unmodelled errors and store its experience and solutions in its memory for future references. It thereby evolves from a system which only has knowledge of limited number of solutions to a system with a complex set of knowledge. This is a relatively unexplored area in autonomous reconfigurable control systems for vehicles.

This paper is organized as follows. The next section describes some of the related work done so far. In section 3 the architecture of the multi agent system is described. Sections 4 and 5 discuss the two stages that need to be integrated. An example is given in section 6 to demonstrate the concept and the results is discussed in section 7. This paper closes with section 8 which concludes the work and details future research to be done.

2 Related Work

A good overview on fault tolerant control systems was presented in [3]. It outlines some of the work done in this field throughout the years. Early work in the integrated design approach was seen in [13][14]. It presented a foundation for an integrated approach to the design of controls and diagnostics in reliable control systems. In this approach the control module and diagnostic module of the control system are designed together, instead of independently, thereby accounting for the interactions which occur between these two modules in a functioning control system. This approach makes use of a generalization of the familiar two parameter controller known as the four parameter controller. This work only dealt with linear systems and left non-linear systems for further research.

A fuzzy model based integration was introduced in [15]. It integrates a model-based adaptive controller with a multi-model fault detection scheme. Four fuzzy models of different sub-processes are used to detect faults. The author left the problem of replacing faulty measurements vulnerable to the performance of an estimation that is subject to further research. Similar work using fuzzy logic can be found in [16][17].

A method which allows one to explicitly incorporate allowable system performance degradation in the event of partial actuator fault in the design process is discussed in [18]. The method is based on model-following and command input management techniques. The degradation in dynamical performance is accounted for through a degraded reference model.

A combination of robust control theory and reinforcement learning to develop a stable neuro-control scheme is presented in [19]. The method of representation for the nonlinear and time varying components of the neural network was to use functional uncertainty. Robust control techniques were applied to guarantee the stability of the neuro-controller. The scheme provides stable control not only for a specific fixed-weight, neural network, but also for a neuro-controller in which weights change during learning. A non-trivial aspect of one of the author's objectives was to develop a suitable learning architecture. The reinforcement learning algorithm was chosen because it is well suited to the type of information available in the control environment.

A more direct approach to reconfigurable control is to use an adaptive systems architecture. In this approach a diagnosis module is omitted and faults are not detected through explicit FDI. It has been quite successfully implemented in aircraft and weapons systems. Work done in this field can be found in [4][10][20].

A relatively new approach to computing is the use of a multi agent architecture in complex software systems. Early work in this field was seen in [21] where "agent oriented programming" has been introduced. A general framework is for instance presented in [22]. Multi agent systems consist of "active software objects" that interact with other components of the system to solve a problems. The author presented how the multi agent framework could be applied to hardware agents. Useful control techniques provided by multi-agent systems for modular, metamorphic robots were demonstrated in [23]. The problem of deadlock, where a situation is reached when all the robots are unable to move or having a set of robots in oscillation was tackled in [24]. To avoid this problem, a general "order" on the environment that guaranteed to build a hierarchy of behaviours between the robots was introduced. All the work above assumed that all the sensors and actuators were working in their nominal mode and did not deal with fault analysis.

A multi agent approach for control systems was proposed in [25]. This architecture contains agents for various components of the system, for example modelling and controller optimization. This new breed of agents are called cautiously optimistic control agents or COCA which applies new modelling results with caution while still using current settings until a certain threshold is exceeded. The key feature is that the central unit does not have full authority over the agents' response. The cooperative action between the different components is what makes the architecture successful.

This work extends the concept in [25] to be used for FDI & RC integration. A multi agent architecture will be used to interface between the two stages. As described in the introduction, machine learning algorithms will be used to equip the agents with a high level of intelligence. It allows the system to dynamically evolve. Not only will it be able to learn unmodelled faults, it will also learn new solutions to deal with the faults. In relation to [23], instead of having multiple agents in one layer, this work will implement a multiple layer structure. Each agent in a different layer has a specific task. It also extends the intelligence of each agent beyond simple rules and logic as described in [23].

3 Architecture

The multi agent system for fault tolerant control will be defined similarly to the ones proposed in [25]. It will have the following agents:

1. Planner agent (A_{p})
2. Controller agent (A_{c})
3. Reconfiguration agent (A_{r})
4. Monitoring agent (A_{m})
5. Worker agent (A_{w})

In general a physical agent A will be a tuple defined by

$$A = \langle \Pi | \Sigma | \Omega \rangle \tag{1}$$

where Π is its physical engine, Σ is its rational behaviour engine and Ω is its continuous engine [30]. Not all three components will be active in particular agents and some may only contain Π or Σ. If present, Ω is a computational unit to compute the future evolution of continuous time models in the environment or in the agents body. Σ is a temporal logic processor that takes assumptions and goals and computes plans to achieve goals under constraints. Π contains access to communication devices and connections to sensors and actuators and can also contain a set of parametrised feedback or feedforward and open loop controllers that enables the agent to interact with its environment via the use of specific groups of the agent's sensors and actuators. This representation method of a basic physical agent serves important practical purposes. It allows for easy programming by engineers, formal verifiability of the system and fast realtime operation on computers [30]. In general, the physical engine Π is a quad tuple defined by

$$\Pi = \langle \gamma | \sigma | \alpha | \Delta \rangle \tag{2}$$

where γ is a set of communication device, σ is a set of sensor data abstractors, α is a set of control data developers and Δ is a set of realtime symbolic feedback loop executors. Any of the Δ, α, σ can be empty. Since γ represents communication devices, it can never be empty. An agent that only has γ defined is called a *software agent*.
The rational behaviour engine Σ is a tuple defined by

$$\Sigma = \langle W | M_{\mathrm{p}} | M_{\mathrm{g}} | C | G | \Re \rangle \tag{3}$$

where W is a world model, M_{p} is an abstract physical skills memory, M_{g} is a goal achievement memory, C is an abstract formulation of behaviour constraints, G is an abstract formulation of short and long term goals and $\Re$ is a reasoning cycle controller [30].

This general implementation platform can fit popular agent architectures well. In the case of Belief Desire Intention (BDI) agents, W can represent the beliefs of an agent, M_{p} can mean the plans of an agent, M_{g} can be associated with relative frequencies (probabilities) of plans succeeding in the past, G could be the goals that it wants to achieve.

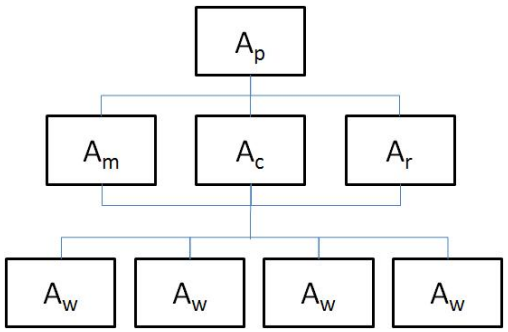


Figure 1. Multi-agent structure

The continuous engine Ω is a tuple defined by

$$\Omega = \langle M | S | O | B | L \rangle \tag{4}$$

where M is a set of approximate continuous models of the world, S is a continuous time simulator that uses analytical and empirical data based dynamical models to predict the future state of the world, O is an optimizer, B is a Boolean evaluator of propositions in terms of σ statements and L is a library of knowledge [30].

The individual agents can then be defined as follows. A_{p} is of the form $\langle \Pi | \Sigma | \Omega \rangle$. All other agents which include $A_{\mathrm{m}}, A_{\mathrm{c}}, A_{\mathrm{r}}, A_{\mathrm{w}}$ are of the form $\langle \Pi | \Sigma \rangle$.

The agents will be implemented in a multilayer structure, where the bottom most layer will be the A_{w} agents dealing with actuators and sensors. At the top, A_{p} will be supervising the overall progress of planning and tasks. A checking of the system status is done via A_{m} which enables unmodelled faults to be taken into consideration. This configuration allows the designer to include prior knowledge of system functionalities.

The reconfiguration of the system is done via A_{r}. For this purpose, reinforcement learning will be implemented for the agent's rational reasoning. This learning algorithm tries to minimize the system error by penalizing actions taken incorrectly and rewarding actions taken correctly.

The Implementation of the changes made to the control structure is done by A_{c}. A diagram of the hierarchy is shown in Fig 1. Each agent will have its own role to play in both the FDI and RC stages.

4 Fault Detection and Isolation (FDI)

To explain how the general agent architecture can be made to work in practice we have chosen to illustrate the principles on an autonomous ground vehicle as shown in Fig. 2. This vehicle has been used to implement a fault tolerant multi-agent control system first in simulation and then the hardware platform of the vehicle.

As mentioned before, the investigation into designing an online reconfigurable controller will be broken down into an FDI stage and a reconfiguration stage. An FDI scheme has three tasks:

1. Fault detection
2. Fault isolation
3. Fault identification

To complete these three tasks, the A_{c}, A_{w} and A_{m} agents work in tandem. A worker agent receives goals from the controller agent A_{c} in the form of velocity v that it must achieve. A_{w} responds by communicating its beliefs and willingness to achieve the goals given by A_{c}. The willingness to cooperate is derived from the belief of how healthy the agent is. A monitoring agent A_{m} counterchecks to see whether the overall goal is achieved. An error is detected when the overall goal stored in M_g of its rational behaviour engine Σ is not achieved within a specified time. If A_{w} is indeed fulfilling its goals, then through the monitoring agent's intelligence, it will communicate what has happened to the planner agent A_{p}. This can only be done if sufficient prior knowledge is given to the agent.

For example, if the A_{c} agent gives a command for a motor velocity of 20 rpm to one of the worker agents A_{w}, then A_{w} first checks its sensors and actuators for its status. This could mean checking for saturation of an actuator or the noise level of a sensor. Based on this information, A_{w} communicates its willingness to cooperate. This could be in the form of informing that it cannot achieve 20 rpm, but is willing to achieve 10 rpm. The controller agent then receives this information and reevaluates its goals.

5 System Reconfiguration (RC)

In this work, a reconfiguration agent A_{r} works with the planner agent A_{p}, monitoring agent A_{m} and controller agent A_{c} to reconfigure the system. When a fault has been detected by the monitoring agent and a hypothesis has been communicated to A_{p}, the planner agent searches its rational behaviour engine for a solution. These limited solutions are known *a priori*. If a solution to the problem is found and it requires that a reconfiguration be executed, the planner agent A_{p} commands the reconfiguration agent A_{r} into effect along with behaviour constraints that A_{r} has to obey.

If on the other hand, the planner agent is not able to find a solution in its rational behaviour engine, it will try to use its continous engine Ω to try and plan a different strategy. Here, inductive learning is implemented. Given a set of rules that it has to abide by, the planner agent implements different strategies to try and achieve its goals. This could be done on a trial and error basis.This is a new philosophy in reconfigurable control. Work done previously only concentrated on limited solutions known *a priori* whereas here the system actively learns new solutions to a problem.

Figure 2. Autonomous Ground Vehicle (AGV)

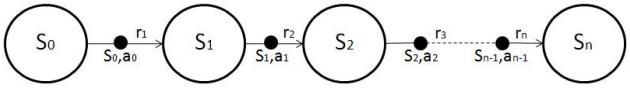


Figure 3. State transition graph

6 Demonstration of Concept

A sample system in the form of an autonomous ground vehicle (AGV) was used to demonstrate the concept of multi-agent architecture. In particular, a reconfiguration agent and a monitoring agent was implemented. Fig. 2 shows the AGV.

In this initial stage, the reconfiguration agent A_r includes the role of worker agents A_w and controller agent A_c. The intelligence of the reconfiguration agent will be in the form of a reinforcement learning algorithm. A brief overview of this algorithm is given in the following.

The idea of reinforcement learning is to try to maximize the amount of reward a learner gets in achieving a goal state from a starting state. It does so by interacting with its environment and taking actions based on these interactions. A reinforcement learning agent will choose previous actions found to be effective in producing large rewards. However, such actions first need to be discovered. Only when actions that it has not selected before is tried can the agent know what rewards it will receive. It has to exploit its knowledge in order to obtain reward, but it also has to explore in order to make better action selections in the future [31]. This characteristic fits in naturally with online reconfigurable control systems where solutions need not be known *a priori*.

A reinforcement learning task can be seen as a Markov decision process, or MDP. Fig. 3 shows how states in a reinforcement learning task are represented.

This scheme works fine for finite set of states. For practical applications however, the number of states are infinite. Fortunately, the states can be generalized using linear function approximation. In our example, gradient-descent Sarsa(λ) algorithm was implemented. It is a vari-

ety of the temporal difference learning method. The action-value function is a linear function of the parameter vector, θ_t. Every state s has a column vector of features ϕ_s. The approximate action-value function is given by

$$Q_t(s) = \theta_t \phi_s = \sum_{i=1}^{n} \theta_t(i)\phi_s(i) \tag{5}$$

Binary features were used because of their simplicity. In particular, the tile coding method was used to divide the state space into partitions [31]. Here, the state space is the velocity and heading angle difference. The action space is the acceleration of the AGV. If the state exists within a partition or tile, then the corresponding feature has the value 1; otherwise the feature value is 0. This simplifies the computation since the updated action-value function Q_t is made by summing up the $\theta_t(i)$ components that correspond to the non-zero features. The update for the action-value function is given by [31]

$$\theta_{t+1} = \theta_t + \alpha\delta_t \mathbf{e}_t \tag{6}$$

α is a learning parameter. δ_t is defined as

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{7}$$

where γ is the discount factor and r_{t+1} is the reward for taking action a_t from state s_t. The eligibility trace $\mathbf{e}_t$ can be viewed as a mechanism to assign blame or credit when an error occurs [31]. It is defined as

$$\mathbf{e}_t = \gamma\lambda \mathbf{e}_{t-1} + \nabla_{\theta_t} Q_t(s_t, a_t) \tag{8}$$

where λ is the blending parameter. The method follows an ϵ-greedy policy for taking actions. This means that the agent takes the action that maximizes its action-value function with a probability of 1 - ϵ. On the other hand, the agent takes a random action with probability ϵ.

In our example, the goal is to have the difference between reference states and actual states to be zero. The state space is divided into 20 tiles and each dimension consists of 200 sections. This value was obtained heuristically. The wheels have a maximum speed of $\pm$90rpm.

The agent has the option of choosing from nine discrete action pairs to transit to a different state. The action pairs are shown in the following table.
τ_{L+} and τ_{R+} denotes positive torque, τ_{L-} and τ_{R-} denotes negative torque and τ_{L0} and τ_{R0}
denotes zero torque. The learner is given 3000 time steps to reach the goal before the episode is terminated and a new episode is started. In our practical example, the tracking problem can be thought of as a finite state MDP. Once the goal state is achieved or the maximum number of steps are taken, the episode ends. A new episode is started and the learning begins again with an improved set of action-values.

In the initial learning stage, a different reference velocity and heading is generated for each episode. A reward

Left torque	Right torque
τ_{L+}	τ_{R+}
τ_{L+}	τ_{R-}
τ_{L+}	τ_{R0}
τ_{L-}	τ_{R+}
τ_{L-}	τ_{R-}
τ_{L-}	τ_{R0}
τ_{L0}	τ_{R+}
τ_{L0}	τ_{R-}
τ_{L0}	τ_{R0}

Table 1. Action pairs

of -10 is given every time the agent takes an action that transitions it into a worse state. This is characterized by two things. One is the difference between reference and actual velocity. The other is the difference between the reference and actual heading angle. If the difference in the current state is bigger than the previous state, the agent is said to have transitioned into a worse state. A reward of 0 is given if the agent takes an action that brings it immediately to its goal. A reward of -1 is given for every other actions. The practice of assigning negative rewards encourages the agent to find a solution in the shortest amount of time. The initial learning process is repeated until the agent's action-values converges to its optimal value Q^*. The model used to train the algorithm is given by [32]

$$\mathbf{M}\ddot{q} + c(q, \dot{q}) = \mathbf{E}(q)\tau \tag{9}$$

where

$$\mathbf{M} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix} \tag{10}$$

$$q = \begin{bmatrix} X \\ Y \\ \theta \end{bmatrix} \tag{11}$$

$$c(q, \dot{q}) = \begin{bmatrix} R_x cos\theta - F_y sin\theta \\ R_x sin\theta - F_y cos\theta \\ M_r \end{bmatrix} \tag{12}$$

$$\mathbf{E}(q) = \begin{bmatrix} cos\theta/r & cos\theta/r \\ sin\theta/r & sin\theta/r \\ c/r & -c/r \end{bmatrix} \tag{13}$$

$$\tau = \begin{bmatrix} \tau_L \\ \tau_R \end{bmatrix} \tag{14}$$

X,Y are the the distance travelled in the respective axes, M_r is the resistive moment acting on the AGV, θ is the heading angle of the AGV, m is the mass of the AGV, I is the identity matrix, R_x is the longitudinal resistive force, r is the wheel radius, $2c$ is the distance between the left and right wheel, and F_y is the lateral resistive force.

Once it has completed its initial learning process, the algorithm can be used on the real system. It has to be

noted that the reinforcement learning algorithm learns continously. This is an advantage for online reconfigurable control systems since the designer has the luxury of instilling the system with prior knowledge. However, when the environment changes or a fault occcurs, the agent uses a trial and error approach to achieve its goals.

When a fault occurs, the reconfiguration agent will keep trying to achieve its goals. It will keep trying even though the goals can never be reached. This could be due to actuator saturation or lack of power. There has to be a method of reassessing its goals after recognizing that the goals cannot be achieved.

A monitoring agent checks to see whether the goals have to be reassessed. It does so by memorizing the time step it takes to achieve a goal for the last k episodes. If the agent's performance dips below a threshold, goal reassessment is done. This is formulated by

$$\hat{p}(\dot{x} \in \dot{x}_{ref} + \xi) = \frac{\sum_{i=n-k}^{n} 1\{\dot{x}_i \in \hat{x}_{ref} + \xi\}}{k} \quad (15)$$

where ξ is a small error value. The monitoring agent then automatically sets the reference velocity to the maximum value that the AGV can achieve.

7 Results and Discussion

In our demonstration, the reinforcement learning algorithm parameters were set as follows: $\lambda = 0.9$, $\alpha = 0.5$,$\gamma = 1$, and $\epsilon = 0$. A total of 2000 episodes were run. The first 2000 episodes were set for the initial learning phase. After the agent's action-value has converged, the algorithm is tested on the AGV with the optimal action-values Q^* .

Fig. 4 shows the initial learning process of the reconfiguration agent. As can be seen, the action value or Q-Value function of the agent converges to its optimal value Q^*, after about 800 episodes. The average timestep taken to achieve its goal is 39 timesteps.

Fig 5 on the other hand, shows the graph of the velocity and heading angle difference against time for different episodes. The performance improves as more and more episodes are executed. The graph shows that when learning starts, the agent takes a very long time to achieve its goal. However, it does manage to reach its goal on the first trial. As the learning progress, it takes only a fraction of the time as it did when it first started learning.

Fig. 6 shows the behaviour of the agent when disturbance is added to the wheels. After the monitoring agent detects that the goal is not achievable, the goal is reassessed to a value that the AGV is able to achieve. From the graph, the agent converges to a solution after only about 50 episodes.

This can be due to a number of factors such as power loss from the battery. The velocity that can be achieved depends highly on the battery usage. Therefore, goal reassessment is needed to consider this loss of power.

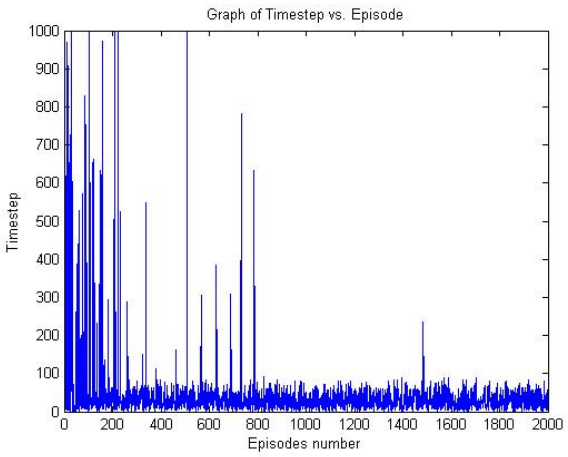


Figure 4. Timesteps taken to reach goal

8 Conclusions and Further Works

A multi-agent architecture to solve the integration of FDI and RC was introduced. Each agent in the architecture was defined and the components belonging the agents were described.

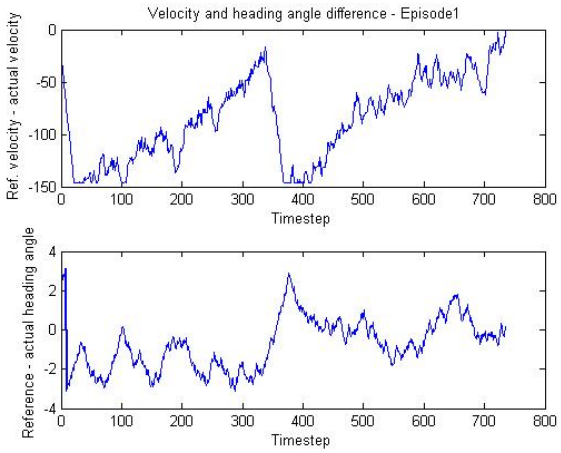
An example was given in sections 6 and 7 to demonstrate how the agent architecture could be defined on practical systems. It was also demonstrated how the agents behave. In the case of reconfiguration agents A_{r}, it tries to adapt to the environment by constantly learning its action values. In the case where the goal cannot be achieved, the monitoring agent detects the fault and goal reassessment is done.

The results of the demonstration showed that the agents were able reassess the its goals after a threshold was crossed. The threshold could be adjusted to fine tune the robustness of the system towards transient faults or pertubations. The results obtained are promising and further work in this direction will be pursued.

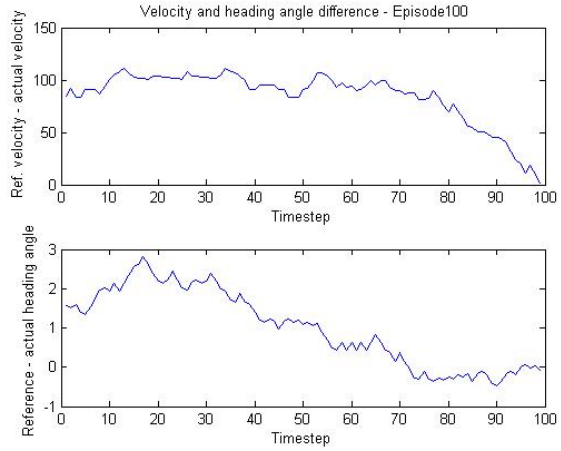
Work done in this research is still at its preliminary stage. Future work will concentrate on incorporating all the other agents in the overall structure. As mentioned in previous sections, machine learning will play a central role in how the agents will interact and successfully control a plant. Various off-the-shelf learning algorithms will be implemented and tested on the system. A comparison will be carried out with respect to their performance. Each algorithm's strengths and weaknesses will be analyzed. This is done by comparing among others its classification error rate, training time, time it takes to make a decision, generalization capability and flexibility. The best algorithm will be selected and implemented as the intelligence of the agents.

References

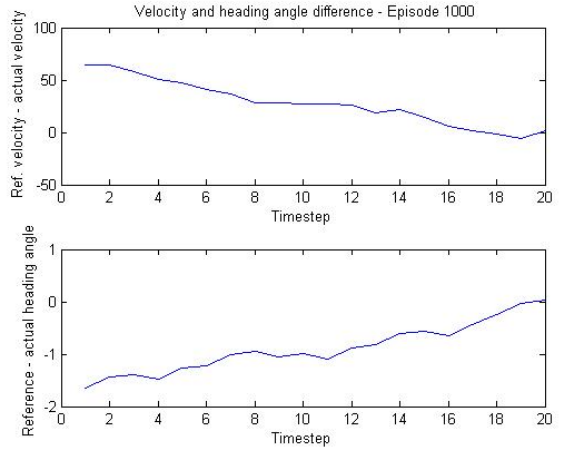
[1] S. Aberkane, J.C. Ponsart, M. Rodrigues, and D. Sauter. *Output feedback control of a class of stochas-*



(a) Episode 1



(b) Episode 100



(c) Episode 1000

Figure 5. Velocity and heading angle difference for different episodes

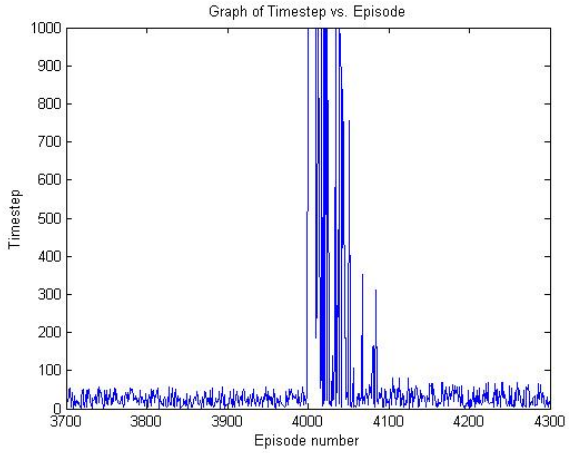


Figure 6. Agent behaviour during actuator saturation

tic hybrid systems. Elsevier automatica, 2008

[2] Halyo N. Broussard J.R. Moerder, D.D. and A.K. Cahlayan. *Application of percomputed control laws in a reconfigurable aircraft flight control system*. Journal of Guidance, Control and Dynamics, 1989.

[3] Youmin Zhang and Jin Jiang. *Bibliographical review on reconfigurable fault tolerant control systems*. Annual Reviews in Control, 32(2):229-252, 2008.

[4] M. Bodson. *Multivariable adaptive algorithms for reconfigurable ight control*. IEEE transactions on control systems technology, vol. 5(no. 2), 1997.

[5] Huibert. Kwakernaak. *Robust control and $\mathcal{H}_\infty$-optimization - tutorial paper*. Automatica, Vol. 29(No. 2):pp. 255-273, 1993.

[6] K.K.T. Thanapalan, S.M. Veres, E. Rogers, and S.B. Gabriel. *Fault tolerant controller design to ensure operational safety in satellite formation flying*. In Decision and Control, 2006 45th IEEE Conference on, pages 1562-1567, Dec. 2006.

[7] K.A. Wise and E. Lavretsky. *Adaptive control of ight: Theory, applications, and open problems*. 2006 American Control Conference 2006: Minneapolis, Minnesota, USA, June 14-16, 2006 2006.

[8] G. Zames. *Feedback and optimal sensitivity: Model reference transformations, multiplicative seminorms, and approximate inverses*. Automatic Control, IEEE Transactions on, 26(2):301-320, 1981.

[9] Brinker J.S. Calise A.J. Enns D.F. Elgersma M.R. Voulgaris P. Wise, K.A. *Direct adaptive reconfigurable flight control for a tailless advanced fighter aircraft*. International journal of robust and nonlinear control. Int. J. Robust nonlinear control, Vol. 9:999-1012, 1999.

[10] D. Shore and M. Bodson. *Flight testing of a reconfigurable control system on an unmanned aircraft*. Proceeding of the 2004 American Control Conference Boston. Massachusetts June 30. July 2,2004, 2004.

[11] J. R. Raol Shobha R. Savanur, Sudesh K. Kashyap. *Adaptive neuro-fuzzy based control surface fault detection and reconfiguration*. In Proceedings of the International Conference on Aerospace Science and Technology, Bangaloire, India, 2008.

[12] D. Shin and Kim Y. Moon, G. *Design of reconfigurable flight control system using adaptive sliding mode control: Actuator fault*. Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering, 219(4):321-328, 2005. 10.1243/095441005X30333.

[13] C. A. Jacobson and C.N. Nett. *An integrated approach to controls and diagnostics using the four parameter controller*. IEEE Contr. Syst. Mag., 11(6), 1991.

[14] N. E. Wu. *Robust feedback design with optimized diagnostic performance*. IEEE transactions on automatic control, Vol. 42(no.9), 1997.

[15] P. Ballé, M. Fischer, D. Fuessel, Onelles, and Risermann. *Integrated control, diagnosis and reconfiguration of a heat exchanger*. IEEE Control systems, 1997.

[16] F. Mrad and G. Deeb. *Experimental comparative analysis of conventional, fuzzy logic,and adapative fuzzy logic controller*. In Industry Applications Conference, Thirty-Fourth IAS Annual Meeting, vol.1, pp.664-673.

[17] A. Tahour, H. Abid, and A.G. Aissaoui. *Adaptive neuro-fuzzy controller of switched reluctance motor*. Serbian journal of electrical engineering, Vol. 4(No. 1):23-34, 2007.

[18] Youmin Zhang and Jin Jiang. *Fault tolerant control system design with explicit consideration of performance degradation*. Aerospace and Electronic Systems, IEEE Transactions on, 39(3):838-848, July 2003. ISSN 0018-9251.

[19] R.M. Kretchmar. *A synthesis of reinforcement learning and robust control theory*. PhD thesis, Department of computer science, Colorado State University, Fort Collins, Colorado, 2000

[20] K.K.T. Thanapalan, S.M. Veres, E. Rogers, and S.B. Gabriel. *Fault tolerant controller design to ensure operational safety in satellite formation flying*. In Decision and Control, 2006 45th IEEE Conference on, pages 1562-1567, Dec. 2006.

[21] Y Shoham. *Agent-oriented programming*. Artificial Intelligence, 60(1):51-92, MAR 1993. ISSN 0004-3702.

[22] H.R. Naji and B.E.Wells. *On incorporating multi agents in combined hardware/software based reconfigurable systems - a general architectural framework*, 2002.

[23] H. Bojinov, A. Casal, and H. Hogg. *Multiagent control of self-reconfigurable robots*. Artifcial Intelligence, Vol. 142:99, 2002.

[24] D. Duhaut, E. Carrillo, and S. Saint-Aime. *Avoiding deadlock in multi-agent systems*. In Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on, pages 1642-1647, Oct. 2007.

[25] S.M. Veres and J. Luo. *A class of bdi agent architectures for autonomous control*. In Decision and Control, 2004. CDC. 43rd IEEE Conference on, volume 5, pages 4746-4751 Vol.5, Dec. 2004.

[26] Argyriou, A., T. Evgeniou, and M. Pontil. *Convex multi-task feature learning*. Machine Learning, 2008. 73(3): p. 243-272.

[27] Zhang, J., Z. Ghahramani, and Y. Yang. *Flexible latent variable models for multi-task learning*. Machine Learning, 2008. 73(3): p. 221-242.

[28] Mehta, N., et al. *Transfer in variable-reward hierarchical reinforcement learning*. Machine Learning, 2008. 73(3): p. 289-312.

[29] Silver, D. and K. Bennett. *Guest editors introduction: special issue on inductive transfer learning*. Machine Learning, 2008. 73(3): p. 215-220.

[30] Veres, S.M, et al. *Rational physical agent: Reasoning beyond logic*.

[31] Sutton, R.S. and A.G. Barto. *Reinforcement learning: an introduction*, ed. J.M. Ockerbloom. 1998, Cambridge, Bradford.

[32] Caracciolo, L., A. de Luca, and S. Iannitti. *Trajectory tracking control of a four-wheel differentially driven mobile robot*. In Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on. 1999.

[33] Kozlowski, K.P. *Modeling and control of a 4-wheel skid-steering mobile robot*. International journal of applied mathematics and computer science, 2004. 14(4): p. 477-496.