6-2012

Self-organizing neural networks for learning air combat maneuvers

Teck-Hou TENG

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

Citation

TENG, Teck-Hou and TAN, Ah-hwee. Self-organizing neural networks for learning air combat maneuvers. (2012). *Proceedings of the International Joint Conference on Neural Networks, Brisbane Australia, June 10-15*. 2859-2866.
Available at: https://ink.library.smu.edu.sg/sis_research/6801

WCCI 2012 IEEE World Congress on Computational Intelligence
June, 10-15, 2012 - Brisbane, Australia IJCNN

Self-organizing neural networks for learning air combat maneuvers

Teck-Hou Teng*, Ah-Hwee Tan*, Yuan-Sin Tan† and Adrian Yeo‡
*School of Computer Engineering, Nanyang Technological University
†DSO National Laboratories
‡CAE Singapore (S.E.A.) Pte. Ltd.

***Abstract*—This paper reports on an agent-oriented approach for the modeling of adaptive doctrine-equipped computer generated force (CGF) using a commercial-grade simulation platform known as CAE STRIVE®CGF. A self-organizing neural network is used for the adaptive CGF to learn and generalize knowledge in an online manner during the simulation. The challenge of defining the state space and action space and the lack of domain knowledge to initialize the adaptive CGF are addressed using the doctrine used to drive the non-adaptive CGF. The doctrine contains a set of specialized knowledge for conducting 1-v-1 dogfights. The hierarchical structure and symbol representation of the propositional rules are incompatible to the self-organizing neural network. Therefore, it has to be flattened and then translated to vector pattern before it can inserted into the self-organizing neural network. The state space and action space are automatically extracted using the flattened doctrine as well. Experiments are conducted using several initial conditions in round robin fashions. The experimental results show that the self-organizing neural network is able to make good use of the domain knowledge with complex knowledge structure to discover the knowledge to out-maneuver the doctrine-driven CGF consistently in an efficient manner.**

I. Introduction

Artificial Intelligence has become a critical component of computer-generated forces (CGFs) typically used for providing training value to the users of the simulators [1]. However, most CGFs that are used in most training simulators depend on non-changing collection of knowledge to provide a variety of behaviors during its interaction with the users. Over time, such CGFs become predictable and become less challenging to the users. In addition, encoding expert knowledge is often very tedious and may even be challenging [2], [3]. Machine learning is recognized as the answer to the limitations of existing rule-based systems [4].

There are different machine learning paradigms for different types of problem [5]. For this work, the air combat problem domain is chosen to investigate real-time learning of air combat maneuvers using self-organizing neural network. Other earlier approaches includes the use of genetic algorithm [6] and artificial neural network [7]. Advancements in UAS [8] and UCAV [9] technologies continue to generate interest in similar domains of application [10].

A scenario of 1-v-1 air combat maneuver is modeled using a commercial simulation platform known as STRIVE®CGF [11]. A Blue CGF and a Red CGF are tasked to out-maneuver each other in a dogfight with the aim of eliminating each other using AIM-9 missiles. In addition, both CGFs are equally capable of launching flares to evade incoming missile. The same air combat maneuver doctrines is available to both CGFs as a basic set of knowledge. But, only the Blue CGF is able to improve on its air combat maneuvering strategies during the dogfights.

This ability to improve air combat maneuvering strategies is achieved using a self-organizing neural network known as FALCON [12]. It is an ART-based neural network [13] that learns incrementally through real-time interactions with the environment. Domain knowledge in the form of a air combat maneuver doctrine is used to automatically extract the state space and action space. And it is also inserted into FALCON to allow it to perform just as well as a doctrine-driven CGF. In addition, it is able to continually discover and refine its air combat maneuvering strategies. Experiments are conducted using several 1-v-1 dogfights scenarios to illustrate this online and incremental learning ability of FALCON. Experimental results shows that it is able to discover new air combat maneuvering strategies that allow it to consistently out-maneuver its adversary.

The rest of the paper is organized as follows. A review of the related works is provided in Section II. Essential details on the FALCON architecture are provided in Section III. This is followed by Section IV where the use of the hierarchical doctrine by the self-organizing neural network is presented. The 1-v-1 air combat maneuver scenario is described in Section V. The experiments and the results to illustrate the use of self-organizing neural network to learn air combat maneuvers are presented in Section VI. The concluding remarks and a brief outlook of future works are provided in Section VII.

II. Related Works

The use of computer generated forces as sparring partners is gaining popularity [14]. Particularly, research on combat games has been ongoing right from the early days [15]. The application of a branch of game theory known as differential game was initially attempted [16]. The use of AI in Air Combat [17] was gradually gaining momentum as artificial neural network has also been applied to air combat domain [7].

TacAir-Soar makes use of the Soar architecture [18] for the modeling of complex knowledge in the air combat missions [19]. However, it is still incapable of expanding its knowledge as available through this work. Subsequent extension of Soar with reinforcement learning was not demonstrated on the same problem domain [20]. Other agent-oriented approaches based on aerial operations include the modeling of fighter combat, airborne early warning and control, stand-off weapon strike missions for F-111 aircraft and the development of tactics and operations for an upgraded AP-3C Orion Maritime Patro Aircraft [10]. Challenges such as the lack of cognitive credibility, the capture of expert knowledge and other human factors are not easily modeled using the agent-oriented approach [21].

TD−FALCON is a class of self-organizing neural networks designed for reinforcement learning in real time [22]. It makes use of a direct code access procedure together with temporal difference learning for handling delayed evaluative feedback. Also considered to be a hybrid model combining the symbolic and connectionist approaches [23], it has a knowledge structure compatible with the generalized modus ponens format that allow for the insertion of domain knowledge [24]. Judges in a recent AI competitions had positive reviews on the performance of FALCON [25].

III. The Reinforcement Learning Model

In this work, the adaptive CGF is driven by a self-organizing neural network known as FALCON. It is based on the adaptive resonance theory (ART), allowing it to incrementally learn and generalize on the vector patterns. Using reinforcement learning, knowledge is discovered during real-time interactions with the environment.

A. FALCON Model and Processes

The FALCON network [12] employs a 3-channel architecture (Fig. 1), comprising a category field F_2^c and three input fields, namely a sensory field F_1^{c1} for representing current states, an action field F_1^{c2} for representing actions, and a reward field F_1^{c3} for representing reinforcement values. A brief summary of the FALCON generic network dynamics, based on fuzzy ART operations [26], is described below.

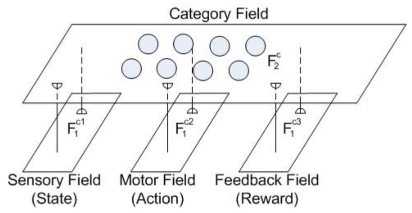


Fig. 1. The FALCON Architecture.

Input vectors: Let $\mathbf{S} = (s_1, s_2, \ldots, s_n)$ denote the state vector, where $s_i \in [0,1]$ indicates the sensory input i. Let $\mathbf{A} = (a_1, a_2, \ldots, a_m)$ denote the action vector, where $a_i \in [0,1]$ indicates a possible action i. Let $\mathbf{R} = (r, \bar{r})$ denote the reward vector, where $r \in [0,1]$ is the reward signal value and $\bar{r}$ (the complement of r) is given by $\bar{r} = 1 - r$. Complement coding is used to normalize the magnitude of the input vectors to prevent the code proliferation problem.
Activity vectors: Let $\mathbf{x}^{ck}$ denote the F_1^{ck} activity vector for $k = 1, \ldots, 3$. Let $\mathbf{y}^c$ denote the F_2^c activity vector. Upon input presentation, $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{A}$, and $\mathbf{x}^{c3} = \mathbf{R}$.
Weight vectors: Let $\mathbf{w}_j^{ck}$ denote the weight vector associated with the jth node in F_2^c for learning the input patterns in F_1^{ck} for $k = 1, \ldots, 3$. Initially, F_2^c contains only one *uncommitted* node and its weight vectors contain all 1's. When an *uncommitted* node is selected to learn an association, it becomes *committed*.
Parameters: The FALCON's dynamics is determined by choice parameters $\alpha^{ck} > 0$ for $k = 1, \ldots, 3$; learning rate parameters $\beta^{ck} \in [0,1]$ for $k = 1, \ldots, 3$; contribution parameters $\gamma^{ck} \in [0,1]$ for $k = 1, \ldots, 3$ where $\sum_{k=1}^{3} \gamma^{ck} = 1$; and vigilance parameters $\rho^{ck} \in [0,1]$ for $k = 1, \ldots, 3$.
Code activation: A bottom-up propagation process first takes place in which the activities (known as choice function values) of the cognitive nodes in the F_2^c field are computed. Specifically, given the activity vectors $\mathbf{x}^{c1}$, $\mathbf{x}^{c2}$ and $\mathbf{x}^{c3}$ (in the input fields F_1^{c1}, F_1^{c2} and F_1^{c3} respectively), for each F_2^c node j, the choice function T_j^c is computed as follows:

$$T_j^c = \sum_{k=1}^{3} \gamma^{ck} \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_j^{ck}|}{\alpha^{ck} + |\mathbf{w}_j^{ck}|}$$

where the fuzzy AND operation $\wedge$ is defined by $(\mathbf{p} \wedge \mathbf{q})_i \equiv min(p_i, q_i)$, and the norm $|.|$ is defined by $|\mathbf{p}| \equiv \sum_i p_i$ for vectors $\mathbf{p}$ and $\mathbf{q}$. In essence, the choice function T_j computes the similarity of the activity vectors with their respective weight vectors of the F_2^c node j with respect to the norm of the weight vectors.
Code competition: A code competition process follows under which the F_2^c node with the highest choice function value is identified. The winner is indexed at J where

$$T_J^c = \max\{T_j^c : \text{for all } F_2^c \text{ node } j\}$$

When a category choice is made at node J, $y_J^c = 1$; and $y_j^c = 0$ for all $j \neq J$. This indicates a winner-take-all strategy.
Template matching: Before node J can be used for learning, a template matching process checks that the weight templates of node J are sufficiently close to their respective activity patterns. Specifically, resonance occurs if for each channel k, the *match function* m_J^{ck} of the chosen node J meets its vigilance criterion:

$$m_J^{ck} = \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck}|}{|\mathbf{x}^{ck}|} \geq \rho^{ck}$$

The match function computes the similarity of the activity and weight vectors with respect to the norm of the activity vectors. Together, the choice and match functions work cooperatively to achieve stable coding and maximize code compression.
When resonance occurs, learning ensues. If any of the vigilance constraints is violated, mismatch reset occurs in which the value of the choice function T_J^c is set to 0 for the duration of the input presentation. The search process then selects another F_2^c node J under the revised vigilance criterion until a resonance is achieved. This search and test process is guaranteed to end as FALCON will either find a *committed* node that satisfies the vigilance criterion or activate an *uncommitted* node which would definitely satisfy the vigilance criterion due to its initial weight values of $1s$.

Template learning: Once a node J is selected, for each channel k, the weight vector $\mathbf{w}_J^{ck}$ is modified by the following learning rule:

$$\mathbf{w}_J^{ck(\text{new})} = (1-\beta^{ck})\mathbf{w}_J^{ck(\text{old})} + \beta^{ck}(\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck(\text{old})})$$

For an uncommitted node J, the learning rates β^{ck} are typically set to 1. For committed nodes, β^{ck} can remain as 1 for fast learning or below 1 for slow learning in a noisy environment. When an uncommitted node is selecting for learning, it becomes *committed* and a new uncommitted node is added to the F_2^c category field.

B. Incorporating Temporal Difference Method

TD-FALCON [22] incorporates Temporal Difference (TD) methods to estimate and learn value functions of state-action pairs $Q(s,a)$ that indicates the goodness for taking a certain action a in a given state s. This is learned as the feedback signal and is used in the selection of the action choices.

As shown in Fig. 2, given the current state s, TD-FALCON first decides between exploration and exploitation by following an action selection policy. For exploration, a random action is picked. For exploitation, TD-FALCON searches for optimal action through a direct code access procedure [27]. Upon receiving a feedback from the environment after performing the action, a TD formula is used to compute a new estimate of the Q-value for performing the chosen action in the current state. The new Q-value is then used as the teaching signal to TD-FALCON to learn the association of the current state and the chosen action to the estimated Q-value.

```
1: Initialize FALCON
2: Sense the environment and formulate a state representation $s$
3: Use *Action Selection Policy* to decide between **Exploration** and **Exploitation**
4: **if** Exploration **then**
5:     Use *Exploration Strategy* to select an action choice from action space
6: **else if** Exploitation **then**
7:     Use *Direct Code Access* to select an action choice from existing knowledge
8: **end if**
9: Use action choice $a$ on state $s$ for state $s'$
10: Evaluate effect of action choice $a$ to derive a reward $r$ from the environment
11: Estimate the $Q$-value function $Q(s,a)$ following a temporal difference formula given by $\Delta Q(s,a) = \alpha TD_{err}$
12: Present state $S$, action $A$ and reward $R$ vectors for **Adaptation**
13: Update the current state $s = s'$
14: Repeat from Step 2 until $s$ is a terminal state
```

Fig. 2. The TD-FALCON Algorithm

Iterative Value Estimation: A value function based on a temporal difference method known as Bounded Q-Learning [28] is used to iteratively estimate the value of applying action choice a to situation s. The estimated Q-value $Q(s,a)$ is learned by TD-FALCON during reinforcement learning. The temporal difference of the value function is iteratively estimated using

$$\Delta Q(s,a) = \alpha TD_{err}(1 - Q(s,a))$$

where $\alpha \in [0,1]$ is the learning parameter, the term $(1 - Q_j(s,a))$ allows the adjustment of Q-values to be self-scaling in such a way that it will not be increased beyond 1.0 and TD_{err} is the temporal error term which is derived using

$$TD_{err} = r + \gamma \max_{a'} Q(s',a') - Q(s,a)$$

where $\gamma \in [0,1]$ is the discount parameter and the $\max_{a'} Q(s',a')$ is the maximum estimated value of the next state s' and r is either the intermediate or terminal reward.

C. Adaptive ϵ-Greedy Action Selection Policy

Using this policy, exploration is performed with a probability of ϵ where $\epsilon \in [0,1]$ [29]. An interval success rate ϕ which is derived using $\phi = \frac{w_s}{w_n}$ where w_s is the number of successful trials within w_n training iterations is used to revise ϵ as $1 - \phi$ after every w_n training iterations.

Subsequently, ϵ is linearly decayed over the next w_n training iterations using an ϵ-decay rate δ which is derived using $\frac{\epsilon}{w_n}$. Such an approach gradually increases exploitation of the learned knowledge within w_n training iterations. This allows it to continually evaluate the effectiveness of the learned knowledge for the situations. The value of ϵ is revised using interval success rate ϕ at every w_n training iterations during reinforcement learning.

IV. Use of Hierarchical Doctrine

The doctrine is a set of specialized knowledge on the execution of air combat maneuvers. It is originating from veteran fighter pilots with combat experience in dogfights. It governs the decision on the choice of responses during air combat. The same doctrine is used to drive the pure rule-based CGF in an existing commercial-grade simulation platform.

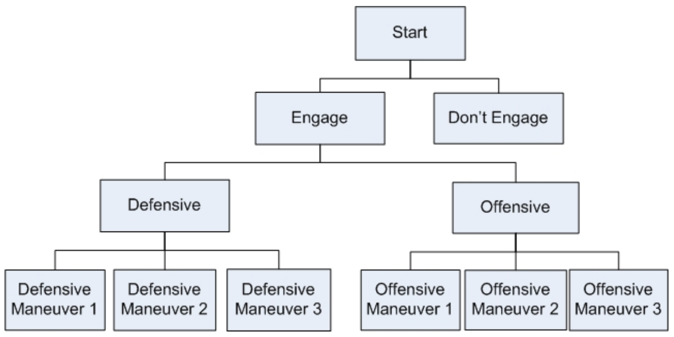


Fig. 3. An illustration of the hierarchical structure of Air Combat Maneuver Doctrine

As illustrated in Fig. 3, it is organized in a hierarchical structure. There is at least one rule bases in the doctrine. And each rule base may have multiple rule sets that are organized hierarchically. Each node in the hierarchical doctrine is a rule set comprised of multiple propositional rules. The hierarchical structure is based on the dependency between the consequent and antecedent of the rule sets. The consequent of a higher level rule set leads to one of the rule sets at the adjacent lower level. The doctrine is available to both CGFs as the domain knowledge for executing 1-v-1 air combat maneuver.

Due to the differences in the knowledge representation scheme and the rule resolution mechanism in TD-FALCON, the hierarchical doctrine will have to be flatten to a single level rule structure and the resultant propositional rules are then translated to vector patterns for insertion into the self-organizing neural network. The flattening of the hierarchical doctrine also allows for the automatic extraction of the state space and action space.

A. Flattening the Hierarchical Doctrine

The hierarchical doctrine is flattened to a single layer structure. The resultant propositional rules are semantically equivalent to the propositional rules in the original doctrine. The general approach involves the linking up of all the antecedents that lead up to the final consequent, i.e., the choice of air combat maneuver. The procedure for the flattening of the hierarchical doctrine is outlined in Fig. 4.

```
1: Form chains of rule sets according to the hierarchical rule
   structure
2: for each rule set within each rule set chain do
3:     Split rules with OR-operator
4:     Link up rules according to their dependency
5:     Apply De Morgan's Law on the flatten rules
6:     Repeat Step 1 - Step 3 for rules with OR-operator
7: end for
8: return Flattened Doctrine
```

Fig. 4. Flattening of Hierarchical Doctrine

The antecedent of the resultant propositional rule is composed of the antecedents of the propositional rules that belong to the same rule set chain. The flattened doctrine will only have propositional rules that directly recommend a particular air combat maneuver. The flattening of the hierarchical doctrine will lead to more propositional rules due to the splitting of propositional rules with the OR-operator. The splitting of such propositional rules is necessary because TD-FALCON represents disjunctive relationship among the propositional symbols as separate propositional rules with the same consequent. In effect, it only accepts propositional rules with conjunctive relationship among the propositional symbols in the antecedent and the consequent.

B. Automatic Extraction of the State Space and Action Space

Defining a reinforcement learning problem for a real world problem domain such as air combat maneuvers requires specialized knowledge. The state space and action space have to be defined as a collection of propositional symbols at the antecedent and consequent respectively. The availability of a well-crafted hierarchical doctrine makes the collection of propositional symbols possible. The hierarchical doctrine is flattened to make this task more convenient.

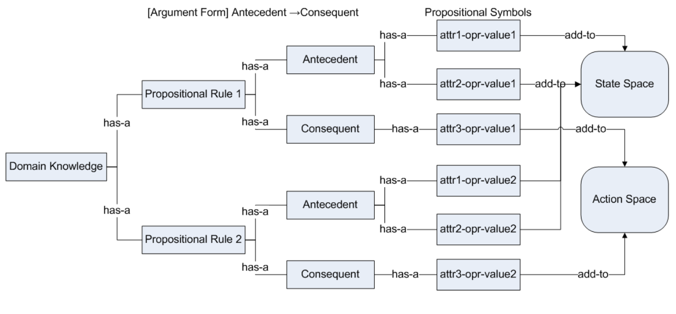


Fig. 5. Extraction of State Space using an existing doctrine

State Space: This is a collection of the propositional symbols from the antecedent of the propositional rules. The propositional symbols are comprised of attributes with different relational operators ($=,<,\leq,>,\geq,\neq$) and different choices of values. With reference to Fig. 5, a complete definition of a propositional symbol is comprised of the name of the attribute, a valid relational operator and a valid value.

The data types and the operational significance of the attribute determines its number of propositional symbols. As illustrated in Fig. 5, the propositional symbols of each attribute in the state space are likely to be distributed over the antecedent of several propositional rules. The flattening of the hierarchical doctrine consolidates all the propositional symbols with conjunctive relationship in a single antecedent. The propositional symbols are then conveniently collected from the antecedent of these resultant propositional rules to form the state space.

The automatic extraction of state space from the doctrine eliminates the complexity of defining the state space manually when doctrine on the problem domain is available. This will ensure the faithful transfer of expert knowledge on the problem domain to the learning model. Additional propositional symbols can still be included to minimize the effect of perceptual aliasing [30].

Action Space: The action space is a collection of propositional symbols for one or more attributes on the responses to the environment. As illustrated in Fig. 5, the action space of reinforcement learning may similarly be automatically extracted using the consequent of propositional rules from the hierarchical doctrine. These may either be the propositional rules at the leaf nodes of the hierarchical doctrine or the resultant propositional rules of the flattening process. In this sense, the flattening of hierarchical doctrine does not has the same benefit to the definition of action space as it does to the definition of the state space.

C. Insertion of Flattened Doctrine

As the knowledge structure of TD-FALCON is compatible with the structure of generalized modus ponens, domain knowledge in the form of propositional rules can be readily inserted into TD-FALCON [24] before learning. Given a rule of the form

IF antecedents THEN consequents FOR reward

the antecedents are translated into a state vector $\mathbf{S}$ and the consequents are translated into an action vector $\mathbf{A}$. The reward vector $\mathbf{R} = (r, 1-r)$, where r is a real value indicating an estimated Q-value of the rule.

The translated state vector $\mathbf{S}$, action vector $\mathbf{A}$ and reward vector $\mathbf{R}$ are then inserted into TD-FALCON using the code activation, code competition, template matching and template learning processes described in Section III-A.

The rule insertion algorithm is summarized in Fig. 6. During rule insertion, the vigilance parameters ρ^{ck} for $k = 1,\ldots,3$ are each set to 1.0 to ensure that only identical set of state, action and reward vectors are grouped into the same

```
1: Initialize TD-FALCON
2: Initialize $\rho^{ck} \leftarrow 1.0$
3: for each propositional rule do
4:     Translate each component of the propositional rule into the vector format
5:         antecedents is translated as state vector $\mathbf{S}$
6:         consequent is translated as action vector $\mathbf{A}$
7:         reward is translated as reward vector $\mathbf{R}$
8:     Present $\mathbf{S}$, $\mathbf{A}$ and $\mathbf{R}$ to TD-FALCON for learning
9: end for
```

Fig. 6. The Rule Insertion Algorithm

recognition node. Thus, each inserted rule leads to a committed cognitive node encoding the $(\mathbf{S}, \mathbf{A}, \mathbf{R})$ tuple as its weight templates. Hence, as many cognitive nodes as the number of inserted rules can be expected. The inserted rules can then be subsequently refined through the temporal difference method during reinforcement learning.

V. 1-v-1 Air Combat Maneuver Scenario

This is based on a classical 1-v-1 pursuit-evasion problem in three-dimensional airspace [31]. There are two CGFs - a Red CGF and a Blue CGF - in the simulated three-dimensional environment (see Fig. 7). Both CGFs are tasked to out-maneuver each other so as to enter into a favorable position to eliminate each other using AIM-9 missiles.

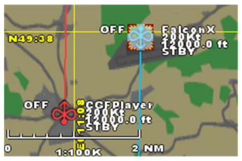
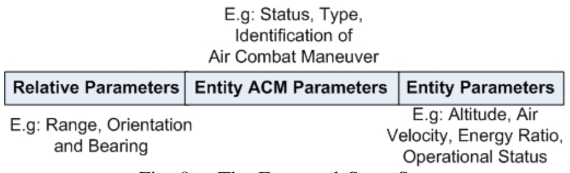


Fig. 7. Illustration of an initial conditions of the Blue CGF (own CGF) and the Red CGF (opponent).

In this case, only the Blue CGF adapts its air combat maneuvers using TD-FALCON. The Red CGF is a doctrine-driven CGF that behaves in a predictable manner. The reinforcement learning problem here is for the Blue CGF to discover the knowledge of using different air combat maneuvers in different situations for the highest possible probability of eliminating the Red CGF in 1-v-1 dogfights.

A. The State Space

The state space of the air combat maneuver problem domain is automatically extracted from the air combat maneuver (ACM) doctrine using the technique outlined in Section IV-B. A total of 15 simple and composite attributes are extracted from the ACM doctrine. A composite attribute is made up of two or more simple attribute such as the composite attribute (mVtt-mVt) is made up of mVtt and mVt simple attribute.



E.g: Status, Type, Identification of Air Combat Maneuver
Relative Parameters
Entity ACM Parameters
Entity Parameters
E.g: Range, Orientation and Bearing
E.g: Altitude, Air Velocity, Energy Ratio, Operational Status

Fig. 8. The Extracted State Space

Given that these attributes are specific to an area of application, it is beyond the scope of this work to elaborate on its significance. With reference to Fig. 8, it may be sufficient to be aware that the extracted state space is comprised of relative parameters such as range and angular position, own entity ACM parameters such as current maneuver, maneuver lock status and own entity parameters such altitude and air speed. These are specified in the form of propositional symbols such as those seen in Table I

TABLE I
Samples of the Propositional Symbols in the State Space

No.	Type	Sample
1	Boolean	wOk = TRUE
2	Float $(>)$	$Abs(mTA) > 135^o$
3	Float $(<)$	$mVtt - mVt < 50$
4	Float $(\leq)$	$mEnergyRatio \leq 0.85$
5	Float $(\geq)$	$Altitude \geq 4500$
6	String $(\neq)$	$mManeuver \neq Weave$

B. The Action Space

The Blue CGF and Red CGF respond to each other using the same set of air combat maneuvers. Both CGFs flies the same of fighter jet so that none of them has the advantage in terms of the execution of the air combat maneuver. In this case, better CGF is the one able to use air combat maneuvers to out-maneuver the other CGF to get into a good position to fire the AIM-9 missile at the opponent.

A total of 13 defensive and offensive air combat maneuvers are available to both CGFs during the simulation. There is one neutral maneuver, three offensive maneuvers and nine defensive maneuvers. The execution of the air combat maneuvers is pre-defined. None of the CGF is able to micro-manage the execution of the air combat maneuvers. The firing of missile and launching of flare are doctrine-driven behaviors. There is no adaptation to evolve the strategies for both behaviors. Therefore, the actions of these two behaviors are not part of the action space of the adaptive component of the CGF.

C. Evaluative Feedback

The evaluative feedback is designed to steer reinforcement learning to achieve the desired effect of the learning task. There are the intermediate reward for the intermediate states and the terminal reward for the terminal states and both of them are derived differently.

Intermediate Reward: The intermediate reward r_i communicates the effect of the chosen response at the intermediate states. Specific reward attributes such as the CGF's proximity to the weapon engagement zone (WEZ) of its opponent (dWEZ), the energy ratio of own CGF to its opponent (eRatio) and the threat level to own CGF with respect to its opponent (tLevel). The normalized derivation of each reward attributes are based on established geometrical formula or heuristics. The intermediate r_i is subsequently derived using the weighted sum approach below.

$$r_i = \omega_{wez}\text{dWEZ} + \omega_{ratio}\text{eRatio} + \omega_{threat}\text{tLevel}$$

where ω_{wez}, ω_{ratio} and ω_{threat} are the weights of the respective reward attributes and the intermediate reward $r_i \in [0.0, 1.0]$. It is used to estimate the value of choosing an air combat maneuver at the intermediate states using the Bounded Q-Learning method.

Terminal Reward: The terminal reward r_t communicates the observed effect at the terminal states to the learning model. The terminal reward function is dependent on the number of observable outcomes at the terminal state. In this case, there are three observable outcomes and the terminal rewards that are used for each of the terminal outcomes are Table II

TABLE II
TERMINAL REWARD FOR AIR COMBAT MANEUVER

No.	Outcome	Terminal Reward
1	Eliminated Red CGF (HasKill)	1.0
2	Time-out of Training iteration (Equal-Match)	0.5
3	Killed by Red CGF (IsKill)	0.0

The allocated value is aimed at giving a sense of desirability to each of these terminal outcomes. Therefore, from Table II, the elimination of the opposing CGF (HasKill) is the best outcome while the elimination of own CGF (IsKill) is the worst outcome at the terminal state. A neutral outcome which is considered to be better than the worst outcome is assigned a value between the best and the worst outcome. It is used to estimate the value of choosing an air combat maneuver at the terminal states using the Bounded Q-Learning method.

VI. EXPERIMENTAL RESULTS

Experiments are conducted to investigate the autonomous learning of air combat maneuvers in 1-v-1 dogfights during iterative real-time interactions. For the experiments, the hierarchical doctrine is inserted into TD-FALCON as the domain knowledge. The other non-adaptive CGF is also driven by the same hierarchical doctrine. Four different initial positions as illustrated in Fig. 9 are used in a round robin fashion in the experiments. This is to demonstrate that TD-FALCON is able to generalize learning to more than one initial conditions within the same session of reinforcement learning.

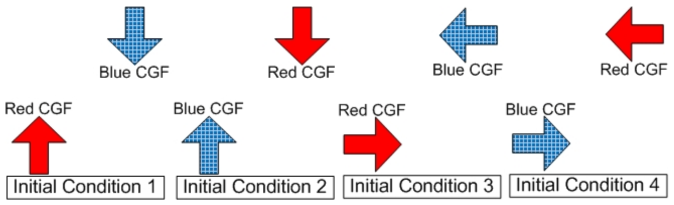


Fig. 9. Illustrations of the four Initial Conditions used for the experiments

The experimental results are collected from a session of reinforcement learning using the four different initial conditions. Table III contains the parameters used for TD-FALCON in these experiments. Every data point in the plots is an average of ten consecutive samples. This gives a total of 12 data points from a reinforcement learning process with 120 training iterations.

A. The Simulation Platform

The simulation is conducted using a suite of commercial-grade simulation software and an own implementation of an inference engine containing the TD-FALCON and the peripheral functions. The simulation is conducted using the server-client configuration. The server is comprised of the suite

TABLE III
PARAMETERS OF TD-FALCON AND ACTION SELECTION POLICY

DA−FACLON Parameters	
Choice Parameters $(\alpha^{c1}, \alpha^{c2}, \alpha^{c3})$	0.1,0.1,0.1
Learning Rates $(\beta^{c1}, \beta^{c2}, \beta^{c3})$	1.0,1.0,1.0
Contribution Parameters $(\gamma^{c1}, \gamma^{c2}, \gamma^{c3})$	0.33,0.33,0.33
Perform Vigilance $(\rho_p^{c1}, \rho_p^{c2}, \rho_p^{c3})$	0.95,0.0,0.45
Learn Vigilance $(\rho_l^{c1}, \rho_l^{c2}, \rho_l^{c3})$	0.0,1.0,0.45
Temporal Difference Learning Parameters	
Learning Rate α	0.5
Discount Factor γ	0.1
Initial Q-Value	0.5
ϵ-Greedy Policy Parameters	
Initial ϵ Value	0.9
ϵ Decay Rate	0.0005
Window Size w_n	10

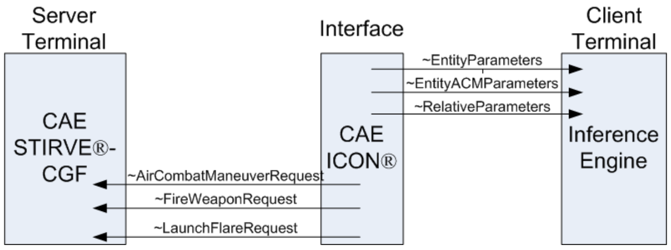
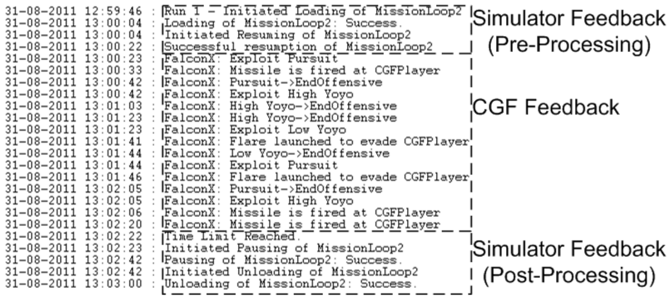


Fig. 10. The Client-Server configuration

of commercial-grade simulation software and the client is our own implementation of the inference engine.

With reference to Fig. 10, the server and client are running on separate terminals. They are physically linked using cross-linked LAN cable and exchanges information via a proprietary software interface known as ICON®. The server presents raw sensory information to the client where it is processed and forwarded to TD-FALCON for action selection and learning.



```
31-08-2011 12:59:46 : Run 1 - Initiated Loading of MissionLoop2
31-08-2011 13:00:04 : Loading of MissionLoop2: Success.
31-08-2011 13:00:04 : Initiated Resuming of MissionLoop2
31-08-2011 13:00:22 : Successful resumption of MissionLoop2
31-08-2011 13:00:23 : FalconX: Exploit Pursuit
31-08-2011 13:00:33 : FalconX: Missile is fired at CGFPlayer
31-08-2011 13:00:42 : FalconX: Pursuit->EndOffensive
31-08-2011 13:00:42 : FalconX: Exploit High Yoyo
31-08-2011 13:01:03 : FalconX: High Yoyo->EndOffensive
31-08-2011 13:01:23 : FalconX: High Yoyo->EndOffensive
31-08-2011 13:01:23 : FalconX: Exploit Low Yoyo
31-08-2011 13:01:41 : FalconX: Flare launched to evade CGFPlayer
31-08-2011 13:01:44 : FalconX: Low Yoyo->EndOffensive
31-08-2011 13:01:44 : FalconX: Exploit Pursuit
31-08-2011 13:01:46 : FalconX: Flare launched to evade CGFPlayer
31-08-2011 13:02:05 : FalconX: Pursuit->EndOffensive
31-08-2011 13:02:05 : FalconX: Exploit High Yoyo
31-08-2011 13:02:06 : FalconX: Missile is fired at CGFPlayer
31-08-2011 13:02:20 : FalconX: Missile is fired at CGFPlayer
31-08-2011 13:02:22 : Time Limit Reached.
31-08-2011 13:02:23 : Initiated Pausing of MissionLoop2
31-08-2011 13:02:42 : Pausing of MissionLoop2: Success.
31-08-2011 13:02:42 : Initiated Unloading of MissionLoop2
31-08-2011 13:03:00 : Unloading of MissionLoop2: Success.
```

Simulator Feedback (Pre-Processing)
CGF Feedback
Simulator Feedback (Post-Processing)

Fig. 11. Real-time updates of the status of the simulation process

The entire simulation routine is controlled using an own implementation of a graphical user interface (GUI) for the inference engine. With reference to Fig. 11, this is also where real-time feedback on the status of the simulation process and also from own CGF are provided The actual execution of the air combat maneuvers by both CGFs can also be observed using the STRIVE®CGF studio manager at the server side.

B. Performance Measures

The performance measures for the air combat maneuver is closely tied to the terminal states. Specifically, the performance measures in Table IV records the number of occurrences of each types of the terminal state during the simulation.

A favorable convergence of learning process is characterized by high percentage of *HasKill* which implies low percent-

TABLE IV
PERFORMANCE MEASURES FOR AIR COMBAT MANEUVER

No.	Terminal State	Descriptions
1	HasKill	Number of times Blue CGF eliminates Red CGF
2	IsKill	Number of times Blue CGF is eliminated by Red CGF
3	EqualMatch	Number of times Blue and Red CGF survive the entire training iteration

age of *IsKill* and *EqualMatch*. However, in situations where elimination of the opponent are difficult to come by, high percentage of *EqualMatch* will be the next best outcome. For the experiments, the 1-v-1 air combat time-out in two minutes.

C. Learning of air combat maneuvers

Plots of the performance measures, the code population and intermediate and terminal rewards are presented in this section. These are the primary performance indicators of the trajectory of the learning process by TD-FALCON in this air combat maneuver problem domain.

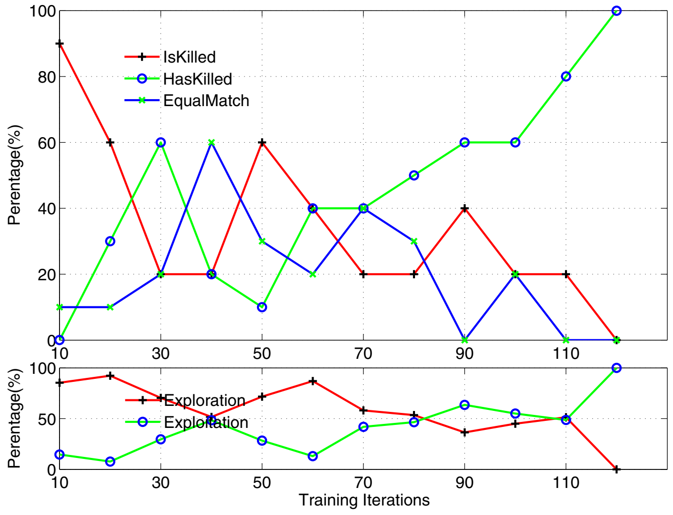


Fig. 12. Top Plot: Plots of the percentage of HasKill, IsKill and EqualMatch. Bottom Plot: Plots of Exploration/Exploitation Rates

With reference to Fig. 12, for the first ten iterations, own CGF has a high rate of IsKill, a small percentage of EqualMatch and was completely incapable of eliminating its opponent. This is correlated to the high rate of explorations at the bottom plot of the same figure. This means that own CGF is in the midst of finding out the maneuvers that are ineffective to the encountered situations. As learning progresses, the upward correlation between the HasKill and the exploitation rates seems to indicate own CGF is able to exploit the learned knowledge effective to it. This upward correlation between these two parameters confirms that learning of own CGF has converged on the knowledge to the four initial conditions in Fig. 9.

Reinforcement learning of TD-FALCON is guided using the feedback signals derived using the reward function. Plots of the intermediate and terminal rewards in Fig. 13 shows that improving evaluation of its intermediate and terminal states. This is primarily correlated to the upward trend of HasKill at the bottom plot. This means using the maneuvers that are effective to the situations lead to own CGF receiving higher reward signals. In turn, this reinforces the selected knowledge to promote its use for subsequent maneuvers.

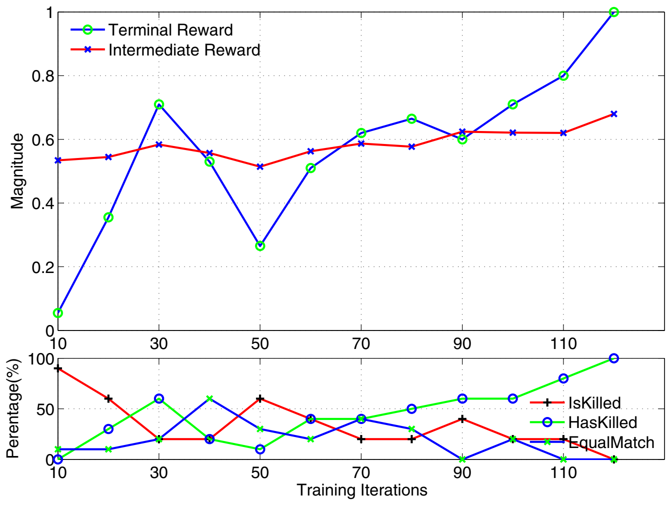


Fig. 13. Top plot: Plots of the Terminal and Intermediate Rewards. Bottom Plot: Plots of Performance Measures.

From Fig. 14, the rate of growth of the node population is correlated to the rate of exploration. High rate of exploration leads to high rate of growth of the node population and vice versa. This shows that TD-FALCON is learning about the effect of different maneuvers for the situations when it is exploring at a high rate. The rate of growth of the node population slows as exploration rates decline. This means there is more updating of the cognitive nodes than the learning of new cognitive nodes.

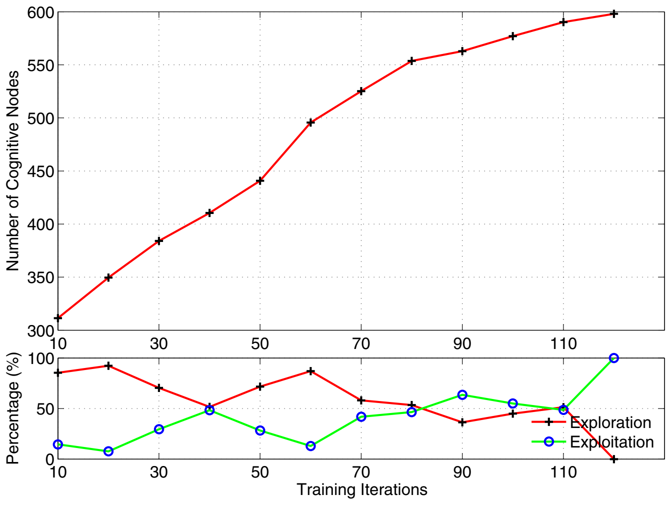


Fig. 14. Top plot: Plot of the node population. Bottom Plot: Plots of Exploration/Exploitation Rates.

Using the adaptive ϵ-greedy action selection policy, the decline in exploration rate is directly attributed to the increase in the interval success rate ϕ. The increase in ϕ is due to the learning of more effective knowledge. Therefore, the availability of effective knowledge increases exploitation rates and thus the rate of growth of node population slows to the point where TD-FALCON is able to fully depend on its learned knowledge at the end of the learning process.

VII. Conclusion

A self-organizing neural network known as TD-FALCON used to evolve 1-v-1 air combat maneuvering strategies in an air combat maneuvering problem has been presented. The learning objective is for the identification of air combat maneuvers that allow the Blue CGF (own CGF) to consistently eliminate the Red CGF in 1-v-1 dogfights. We have flattened a hierarchical doctrine to create an equivalent doctrine with a flat structure. This is used to automatically extract the state space and action space. The resultant propositional rules are also translated and inserted into TD-FALCON as the domain knowledge to bootstrap reinforcement learning.

The various aspects of the response characteristics of TD-FALCON are illustrated using plots of the performance measures such as HasKill, IsKill and EqualMatch, the reward signals and the node population. Experimental results have shown TD-FALCON to be capable of converging to favorable outcomes for four initial conditions that are used in a round robin fashion. Specifically, the performance plots are able to show own CGFs learning towards positive outcomes. The reward plots illustrated the correlation between the rewards and the learning outcomes. The node population plot shows stabilizing of the node population as learning converges.

TD-FALCON was already compared with the other existing approaches using other platforms [22], [25]. Comparison with the alternative approaches is not included here because there is no alternative implementation of adaptive CGF in this commercial-grade simulation platform. Going forward, there are plans to use TD-FALCON integrated into a modular and distributed cognitive architecture (CA) [32] to work with a dynamic reasoner [33] for modeling of CGFs in team-oriented air combat missions [34].

Acknowledgment

This work is supported by the DSO National Laboratories under research grant DSOCL11258. This project was conducted in close collaboration with Khee-Yin How and his team at DSO National Laboratories, Seng-Beng Ho and his team at Temasek Laboratories@NUS, Adrian Yeo and his team at CAE (S.E.A.) Pte. Ltd. and Sylvain Caron and his team at CAE (Montreal) Inc.

References

[1] G. Fong, "Adapting COTS games for military experimentation," *Simulation and Gaming*, vol. 37, no. 4, pp. 452–465, 2006.

[2] R. H. Ashton, "Combining the judgments of experts: How many and which ones?," *Organizational Behavior and Human Decision Processes*, vol. 38, pp. 405–414, 1986.

[3] G. Wright and P. Ayton, "Eliciting and modelling expert knowledge," *Decision Support Systems*, vol. 3, pp. 13–26, 1987.

[4] L. Galway, D. Charles, and M. Black, "Machine learning in digital games: a survey," *Artificial Intelligence Review*, vol. 29, pp. 123–161, 2008.

[5] T. Mitchell, *Machine Learning*, McGraw Hill, 1997.

[6] J. J. Grefenstette, C. L. Ramsey, and A. C. Schultz, "Learning Sequential Decision Rules Using Simulation Models and Competition," *Machine Learning*, vol. 4, pp. 355–381, 1990.

[7] E. Y. Rodin and S. M. Amin, "Maneuver prediction in air combat via artificial neural networks," *Computers & Mathematics with Applications*, vol. 24, no. 3, pp. 95 – 112, 1992.

[8] R. S. Stansbury, M. A. Vyas, and T. A. Wilson, "A survey of UAS Technologies for Command, Control and Communication (C3)," *Journal of Intelligent Robot System*, vol. 54, pp. 61–78, 2009.

[9] The Economist, "Flight of the drones: Unmanned aerial warfare," *The Economist*, vol. 401, no. 8754, pp. 30–34, 2011.

[10] C. Heinze, M. Papasimeon, S. Goss, M. Cross, and R. Connell, "Simulating fighter pilots," in *Defence Industry Applications of Autonomous Agents and Multi-Agent Systems*, pp. 113–130. 2007.

[11] CAE Inc., "CAE STRIVE®CGF," Product Brochure, 2007.

[12] A.-H. Tan, "FALCON: A Fusion Architecture for Learning, Cognition, and Navigation," in *Proceedings of the IJCNN*, 2004, pp. 3297–3302.

[13] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, and Image Processing*, pp. 54–115, 1987.

[14] R. E. Wray, J. E. Laird, A. Nuxoll, D. Stokes, and A. Kerfoot, "Synthetic Adversaries for Urban Combat Training," *AI Magazine*, vol. 26, no. 3, pp. 82–92, 2005.

[15] M. D. Ardema, M. Heymann, and N. Rajan, "Combat Games," *Journal of Optimization Theory and Applications*, vol. 46, no. 4, pp. 391–398, 1985.

[16] W. Grimm and K. H. Well, *Lecture Notes in Control and Information Sciences*, chapter Modeling Air Combat as Differential Game: Recent Approaches and Future Requirements, pp. 1–13, 1991.

[17] E. Y. Rodin, Y. Lirov, S. Mittnik, B. G. McElhaney, and L. Wilbur, "Artificial intelligence in air combat games," *Computers & Mathematics with Applications*, vol. 13, no. 1-3, pp. 261 – 274, 1987.

[18] John E. Laird, Allen Newell, and Paul S. Rosenbloom, "Soar: An architecture for general intelligence," *Artificial Intelligence*, vol. 33, no. 1, pp. 1 – 64, 1987.

[19] R. M. Jones, J. E. Laird, P. E. Nielsen, K. J. Coulter, P. Kenny, and F. V. Koss, "Automated intelligent pilots for combat flight simulation," *AI Magazine*, vol. 20, no. 1, pp. 27–42, 1999.

[20] S. Nason and J.E. Laird, "Soar-RL:integrating reinforcement learning with soar," *Cognitive Systems Research*, vol. 6, pp. 51–59, 2005.

[21] C. Heinze, B. Smith, and M. Cross, "Thinking quickly: Agents for modeling air warfare," in *Advanced Topics in Artificial Intelligence*, vol. 1502, pp. 47–58. 1998.

[22] A.-H. Tan, N. Lu, and X. Dan, "Integrating Temporal Difference Methods and Self-Organizing Neural Networks for Reinforcement Learning with Delayed Evaluative Feedback," *IEEE Transactions on Neural Networks*, vol. 19, no. 2, pp. 230–244, February 2008.

[23] H.-Q. Chong, A.-H. Tan, and G.-W. Ng, "Integrated cognitive architectures: A survey," *Artificial Intelligence Review*, vol. 28, no. 2, pp. 103–130, 2007.

[24] T.-H. Teng, Z.-M. Tan, and A.-H. Tan, "Self-organizing neural models integrating rules and reinforcement learning," in *Proceedings of the IJCNN*, June 2008, pp. 3770–3777.

[25] D. Wang, B. Subagdja, A.-H. Tan, and G.-W. Ng, "Creating human-like autonomous players in real-time first person shooter computer games," in *Proceedings of the* 21^{st} *Annual Conference on Innovative Applications of Artificial Intelligence*, 2009, pp. 173–178.

[26] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, vol. 4, pp. 759–771, 1991.

[27] A.-H. Tan, "Direct Code Access in Self-Organizing Neural Networks for Reinforcement Learning," in *Proceedings of the IJCAI*, January 2007, pp. 1071–1076.

[28] C. J. C. H. Watkins and P. Dayan, "Q-Learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[29] A. Pérez-Uribe, *Structure-Adaptable Digital Neural Networks*, Ph.D. thesis, Swiss Federal Institute of Technology-Lausanne, 2002.

[30] L. Chrisman, "Reinforcement learning with perceptual aliasing: the perceptual distinctions approach," in *Proceedings of the National Conference on Artificial Intelligence*, 1992, pp. 183–188.

[31] M. D. Ardema and N. Rajan, "An approach to three-dimensional aircraft pursuit-evasion," *Computers & Mathematics with Applications*, vol. 13, no. 1-3, pp. 97–110, 1987.

[32] G.-W. Ng, Y.-S. Tan, L.-N. Teow, K.-H. Ng, K.-H. Tan, and R.-Z. Chan, "A cognitive architecture for knowledge exploitation," in *Proceedings of the* 3^{rd} *Conference on Artificial General Intelligence*, 2010.

[33] G.-W. Ng, K.-H. Ng, K.-H. Tan, R.-Z. Chan, and C.-H.-K. Goh, "Novel methods for fusing bayesian knowledge fragments in d'brain," in *Proceedings of the* 10^{th} *International Conference on Information Fusion*, 2007, pp. 1–8.

[34] K. Virtanen, R. P. Hämäläinen, and V. Mattila, "Team Optimal Signaling Strategies in Air Combat," *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, vol. 36, no. 4, pp. 643–660, 2006.