Integrated Hybrid Planning and Programmed Control for Real–Time UAV Maneuvering

Miquel Ramirez
The University of Melbourne
miguel.ramirez@unimelb.edu.au

Michael Papasimeon*
The University of Melbourne
michael.papasimeon@unimelb.edu.au

Nir Lipovetzky
The University of Melbourne
nir.lipovetzky@unimelb.edu.au

Lyndon Benke†
The University of Melbourne
lbenke@student.unimelb.edu.au

Tim Miller
The University of Melbourne
tmiller@unimelb.edu.au

Adrian R. Pearce
The University of Melbourne
adrianrp@unimelb.edu.au

Enrico Scala
Foundation Bruno Kessler
escala@fbk.eu

Mohammad Zamani
The University of Melbourne
mohammad.zamani@unimelb.edu.au

ABSTRACT

The automatic generation of realistic behaviour such as tactical intercepts for Unmanned Aerial Vehicles (UAV) in air combat is a challenging problem. State-of-the-art solutions propose hand–crafted algorithms and heuristics whose performance depends heavily on the initial conditions and aerodynamic properties of the UAVs involved. This paper shows how to employ domain–independent planners, embedded into professional multi–agent simulations, to implement two–level Model Predictive Control (MPC) hybrid control systems for simulated UAVs. We compare the performance of controllers using planners with others based on behaviour trees that implement real world tactics. Our results indicate that hybrid planners derive novel and effective tactics from first principles inherent to the dynamical constraints UAVs are subject to.

KEYWORDS

planning; agent programming; hybrid systems; UAV

ACM Reference Format:
Miquel Ramirez, Michael Papasimeon, Nir Lipovetzky, Lyndon Benke, Tim Miller, Adrian R. Pearce, Enrico Scala, and Mohammad Zamani. 2018. Integrated Hybrid Planning and Programmed Control for Real–Time UAV Maneuvering. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

In computational operations research (OR), multi-agent simulations (MAS) are often used to model, analyse and understand complex socio-technical systems [23]. In the defence domain, such simulations are used to support the acquisition of new aircraft, to evaluate system upgrades, to assess tactical behaviour [24, 48] and to explore future operational concepts such as employment of autonomous systems [7]. Multi-agent simulations of air combat are challenging due to both the highly dynamic and adversarial nature of the domain and the complexity in the systems and the team tactics being modelled. These challenges manifest themselves across the entire spectrum of the software engineering and operational analysis processes, from specifying complex team tactical behaviour [12, 22], up to representing these complex behaviours within agent reasoning frameworks for verification and validation.

Model Predictive Control (MPC) refers to a range of control methods, rather than a specific control strategy, which make explicit use of *models* of processes — aircraft dynamics in our case – to obtain a mixed discrete–continuous time–varying control signal that minimises a given objective function [5, 9]. While MPC is a general framework, most existing approaches have difficulties dealing with systems where the constraints on dynamics are other than linear or change over time, and require substantial application-specific engineering in order to be applicable or scale up [8]. The *Domain Predictive Control* (DPC) framework [32] presents domain–independent planning as an alternative, to systematise heuristic solutions [43] to these challenges [26]. Like MPC, DPC uses an explicit model to predict future states, but instead of relying on ad–hoc descriptions of states and transitions, these are compactly described by means of a domain *theory* given in a formal abstract language. We reformulate Löhr's DPC framework over an extension of the FSTRIPS [17] planning language that includes some of PDDL+ [14] features. This allows to represent arbitrary hybrid dynamical systems [20] *directly*, without requiring their discrete–time solution, and is expressive enough to account *both* for arbitrary *actor models* [28] and procedural control knowledge [10].

In this paper we discuss a novel pilot agent for a challenging adversarial task, *stern conversion* [1, 46] in which two UAVs, Blue and Red from now on, compete to get behind each other. This is a complex domain which allows to illustrate the potential of DPC augmented with state–of–the–art planning languages [15] and search algorithms [16, 30, 31]. Section 2 describes the task, which requires some form of sub-goaling to be solved, and briefly sketch the simulation environment used in Section 3.1. Section 4 discusses how MPC controllers can be implemented with hybrid planners, and

[*] Michael Papasimeon is seconded to the University of Melbourne from the Australian Defence Science and Technology (DST) Group.

[†] Lyndon Benke is a PhD student and is from the Australian Defence Science and Technology (DST) Group.

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

the semantics of the plans computed. We describe the dynamics and constraints posed by a straightforward supervisory control scheme that the planner uses directly in Section 5. Next we describe the experimental setting and discuss empirical observations on the performance of planning-based controllers that integrate procedural control in several different ways, with an implementation of Shaw's heuristic [46] for stern conversion. Our results show that the planner reveals tactics automatically from first principles, that sometimes outperform those taught to real world pilots.

2 AERIAL INTERCEPTS

Relative maneuvering between aircraft typically falls into two categories; Beyond Visual Range (BVR) and Within Visual Range (WVR) maneuvering. Tactics for BVR maneuvering make use of long range sensors such as radar, whereas WVR tactics rely on shorter range ones such as pilot eyesight, and optical or infra-red cameras. The shorter relative distances in WVR maneuvering make it more difficult for a pilot, whether human or an autonomous agent, to implement these tactical maneuvers in a robust and *timely* manner.

In air operations, the term *intercept* refers to maneuvering an aerial vehicle to a desired position and orientation relative to another aircraft in order to enable formation flying, the use of sensors for identification or, in defence scenarios, the engagement of weapon systems. The two most common types of aerial intercepts are forward and rear quarter intercepts [46], the latter involve maneuvering the aircraft to be behind the vehicle being intercepted.

The most common way of achieving a rear quarter intercept is through a *stern conversion* maneuver. While a *stern conversion* can be achieved in many different ways, the most common approach, or *tactic*, is described in Fig. 1 which is adapted from Shaw's treatise [46] on fighter combat. For the purposes of this paper we will refer to this *maneuver plan* as Shaw's heuristic (SH). The Figure shows a schematic of Blue executing Shaw's heuristic under idealised conditions. In this scenario, Red flies straight and level. At time t_0, Blue begins to execute the stern conversion maneuver against Red at a *separation* or *turn* range Δx. At this point Blue starts adjusting its initial heading to ϕ in order to achieve a *lateral separation* or *displacement* of Δy w.r.t. Red. Once the desired Δy has been achieved at time t_1, Blue adjusts back its heading from ϕ. At this point Blue is flying parallel to and in the opposite direction as Red. This phase of the maneuver is referred to as *flying reciprocal*. At some later time t_2, the distance between Blue and Red becomes less than or equal than what is known as the *conversion range* r. This is the range at which Blue is to turn into Red, and eventually position itself behind it. The value of r to use depends on the relative velocities Red and Blue as well as Blue's turning capabilities that follow from the aerodynamic properties of its airframe.

The specific way in which it is determined if Blue has successfully achieved a rear quarter intercept usually depends on the context for the intercept, but some common patterns can be identified. Typically, the intercept is deemed as achieved whenever Blue maintains its position *behind* Red for a given period of time, and subject to a number of additional constraints. Figure 2 shows a diagram demonstrating such constraints, which we will later formalise in Section 6. Informally, we will consider the intercept to be achieved whenever, besides the constraint on the *antenna train angle* [46] ATA between Blue and Red, constraints on range, altitude differences and speeds are also upheld for 5 seconds. In addition to the intercept success criteria, Figure 2 also defines the relative aspect and antenna train angles between the two UAVs. R_{br} is the range (in meters) between Blue and Red, R_{min} (R_{max}) is the minimum (maximum) intercept range, ϕ_{max} is the maximum intercept angle, *AA* is the aspect angle between Blue and Red [35] and $\vec{V}_b$ and $\vec{V}_r$ are the velocity vectors of Blue and Red respectively.

3 ACE MULTIAGENT SIMULATION

The MAS environment used in this paper is ACE (Air Combat Environment), a team-oriented MAS framework currently under development by the Australian Defence Science and Technology (DST) Group [33, 34]. ACE is designed to simulate teams of aircraft in adversarial n-versus-m air combat missions. These simulations are used in operations research studies to support the acquisition of new aerospace systems and to explore how to best employ them [38].

The scenario we consider in this paper consists of a blue and a red flight [1] each consisting of a single entity representing a UAV. Each UAV entity consists of three components, modelling the UAV flight dynamics, sensors and the last representing decision making. We refer to this as the *pilot agent* component. Represented in the simulated aircraft is also the state of the UAV's primary sensor which includes the information on the orientation of the sensor array and the list of entities the sensor has detected and could track. We leave the sensor component out of the discussion as we will be assuming perfect observability of the quantities describing other aircraft states. We briefly describe the first and third next.

3.1 Flight Dynamics

We have used a simplified model of flight dynamics to keep run–times reasonable yet still provide a good approximation of rear quarter aerial intercepts, that is, the trade–off between speed and turn rate. There are two components to the simulation model. Namely, a time-stepped simulation representing simplified flight physics and a basic control system that allows the pilot agent to request the flight dynamics model to undertake specific maneuvers. Flight model state variables include the UAV's position (x, y, z), its orientation (ψ, θ, ϕ), its speed and G-load factor. The orientation of the UAV is represented by the Euler angles (ψ, θ, ϕ) corresponding to rotations around the z, y and x axes. These are typically referred to as the *yaw* (or heading, in global coordinates), *pitch* and *bank* angles [25]. The model also maintains a reciprocal set of state variables known as the *command variables*: x_c, y_c, z_c, ψ_c, θ_c, ϕ_c and $gload_c$. These are commanded or desired values issued to the UAV by the pilot agent, for example, z_c is the desired or commanded altitude. A simple feedback control system [28] takes these values and uses them to steer the simulated aircraft. The equations of motion for the flight dynamics make use of a simplified physics model and are shown in Equations 1-2 below

$$\dot{x} = v(t)\cos\psi(t), \qquad \dot{y} = v(t)\sin\psi(t), \qquad \dot{z} = v(t)\sin\theta(t) \tag{1}$$

$$\dot{\psi} = g\,\frac{\tan\phi(t)}{v(t)} \qquad \dot{\phi} = \epsilon\phi_k \tag{2}$$

[1] A flight is the terminology used to represent a team or formation of aircraft.

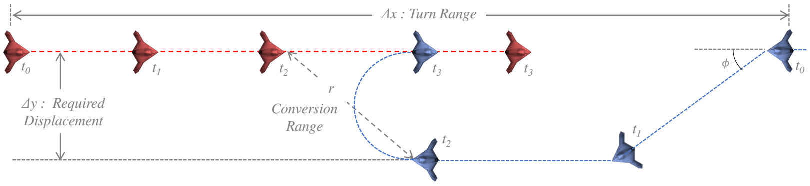

Figure 1: A schematic for a blue UAV undertaking a stern conversion maneuvre against a red UAV.

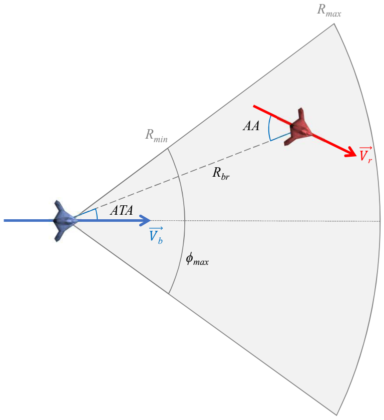

Figure 2: Intercept success criteria and relative angle definitions between Blue and Red.

While the differential equations above do not account for *lift forces* [25], the trade–off between speed and velocity is captured via Equation 2. $\dot{\psi}$ is measured in rad/s and depends on current speed $v(t)$ and bank angle $\phi(t)$, and Earth's gravitational constant g. The bank angle rate of change $\dot{\phi}$ depends on $\epsilon = \text{sgn}(\phi_c - \phi(t))$, the sign of difference between commanded and actual bank angle, and fixed parameter ϕ_k.

3.2 Pilot Agent

Four parametric commands are available to the pilot agent to control the UAV. These are sent by the pilot agent to the flight control system which then interprets them and sets low–level control signals $\dot{\psi}(t)$, $\dot{\theta}(t)$, $\dot{\phi}(t)$ and $\dot{v}(t)$ accordingly. `SetFlyLevelCmd` () takes no parameters and sets all control signals to zero. `SetPitchAngleCmd` (θ_c) and `SetSpeedCmd` (v_c) set the inputs of straightforward feedback control systems to $\dot{\theta}$ and $\dot{v}$ respectively. `SetHeadingGLoadCmd` allows the pilot to specify a desired heading ψ_c as well as a desired g-load factor $gload_c$. This latter parameter limits the magnitude of centripetal forces allowed during the turn to achieve a change in heading.

As a baseline for the experiments discussed in Section 6, the heuristic depicted in Figure 1 was implemented in ACE as a behaviour tree (BT) [10]. Figure 3 shows a graphical representation of the BT for Shaw's heuristic. Rectangular leaf nodes represent actions, rounded corner nodes stand for *conditions*. Condition and action nodes are composed hierarchically with internal square nodes accounting for parallel execution (=), totally–ordered sequencing (▶) or fallbacks (?). The behaviour tree was implemented using the open source BT++ library [10]. In describing Figure 3, we make reference to the Shaw Heuristic schematic for a stern conversion in Figure 1 and accompanying discussion in Section 2. The execution of the behaviour tree starts with a parallel node with three children. The rationale for this is that Blue is always trying to match the speed and altitude of its opponent via the *Match Speed* and *Match Altitude* actions. At the same time, and for each time step, it is necessary to identify in which stage of the plan depicted in Figure 1 the UAV is. For this we recursively nest fallback nodes, each checking for the conditions that enable the behaviours necessary to achieve the *sub–goals* required by the plan. The *Pure Pursuit* behaviour, equivalent to *Convert*, is used to steer Blue towards Red when the range is greater than R_{sc}. The other three behaviours implement the turns at times t_0, t_1 and t_2, generating the low-level control signals that steer the aircraft as required.

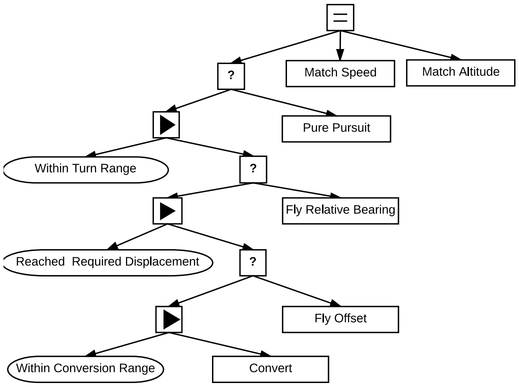

Figure 3: Behaviour tree implementing Shaw's heuristic.

4 DOMAIN PREDICTIVE CONTROL FOR HYBRID SYSTEMS

Model Predictive Control (MPC) [9] is based on the iterative receding horizon solution of a finite horizon optimal control problem formulated on a *model* of the system dynamics, plant, control constraints, and performance objective. Domain Predictive Control (DPC) [32] instances MPC so that *planning models* are used to represent dynamics and constraints, and *planners* to solve the optimal control problem or find an approximation. In this Section we give a formal definition of two–level MPC controllers that integrate *supervisory control* [28] and the generation of low-level control signals to steer in a timely fashion the simulated vehicles.

4.1 Receding Horizon Control

We next formalise two–level MPC controllers for hybrid systems by putting together Borrelli's [5] discussion of Receding Horizon Control (RHC) in such a setting, and Di Cairano's [8] discussion of MPC controllers in industrial settings. At any control cycle[2] t the MPC controller will perform three steps, which are repeated. First, a *finite* horizon optimal control problem is set, that uses the current state $x(t)$ as the *initial* state. Second, the control problem is solved obtaining the optimal input sequence $U_{t\to t+N|t} = \{u_0, u_1, \ldots, u_k, \ldots, u_{N-1}\}$ over the horizon N. Third and last step is to apply the computed optimal sequence until a new state $x(t+1)$ becomes available. In ACE simulations, the controller receives states at a fixed constant rate set. The finite time optimal control problem set is given by the following equations

$$\min_{U_{t\to t+N|t}} J\Big(x(t)\Big) \tag{3}$$

subj. to

$$x_{k+1} = f_i(x_k, u_k), \text{ if } \begin{bmatrix} x_k \\ u_k \end{bmatrix} \in \mathcal{M}_i \tag{4}$$

$$h_C(x_k, u_k) \geq 0 \tag{5}$$

$$x_N \in \mathcal{X}_f \tag{6}$$

$$x_0 = x(t) \tag{7}$$

where t is the discrete index of the control cycle. $x_k \in \mathbb{R}^n$ and $u_k \in \mathbb{R}^m$ are the values of states and input signals predicted k steps ahead of t, based on information up to t. $h(\cdot)$ is positive whenever states and input signals satisfy a given set of global constraints C. Pairs $(\mathcal{M}_i, f_i)$ characterise the possible *modes* of the hybrid system, $\mathcal{M}_i$ being a *partition* of the combined state and input space $\mathbb{R}^{n+m}$. f_i is a *state transition function* that maps states x_k and inputs u_k into future predicted states x_{k+1}. $\mathcal{X}_f$ is the set of *accepted terminal* states of the system, further constraining possible trajectories throughout the state space. These are then ranked according to *cost functions* J

$$J\Big(x_0\Big) = p\Big(x_N\Big) + \sum_{k=0}^{N-1} q\Big(x_k, u_k\Big) \tag{8}$$

where $p(\cdot)$ and $q(\cdot)$ are respectively the *terminal* and *stage* cost functions. As observed by Di Cairano, from Equations 3-7 MPC results in a nonlinear static state–feedback controller

$$u(t) = g_{MPC}(x(t)) \tag{9}$$

since at every control cycle the only changing element in Eq. 3-7 is the initial state $x(t)$. When analytical solutions to Eq. 9 exist, the control algorithm becomes very simple, as it only needs to evaluate the formula of the solution on $x(t)$. Usually, and as it is the case of the systems considered in this paper, the explicit feedback law in Eq. 9 is *impossible* to compute exactly [3]. Many methods have been proposed to compute approximate solutions to Eq. 3-7, like restricting modes dynamics f_i to be linear–time invariant (LTI) systems confined to partitions of $\mathbb{R}^{n+m}$ which are polyhedra, and cost functions to be given as linear or quadratic functions of states and inputs [5]. Such *problem approximations* [4] enable the use of powerful mathematical programming solvers off-the-shelf [43]. Alternatively, the so–called *recursive* approach to Eq. 3-7, formulates a dynamic–programming task which can be solved in a variety of forms. An example of this approach being used for the intercepts discussed in Section 2 is the work of McGrew et al [35], that use real–time dynamic programming (RTDP) [2] to obtain estimates of $J^*\Big(x_k, u_k\Big)$, the *optimal solution* to Eq. 3-7. These estimates are then used by a *rollout algorithm* implementing a one-step lookahead that selects input signals and realise g_{MPC}. In the literature on AI planning these strategies are usually referred to as *re-planning* or *on-line planning*, as action selection and execution are interleaved, solving a planning task at each time step [19].

4.2 DPC over Functional STRIPS

Löhr et al. [32] first observed that hybrid control systems could be specified as planning domains, with actions accounting for voluntary *switching* between modes $(\mathcal{M}_i, f_i)$. Plans then describe *implicitly* the continuous time evolution of *time switched* closed–loop hybrid control systems [29]. We will lift the requirement to linearise f_i and avoid the combinatorial explosion inherent to enumerating control modes, thus significantly broadening the applicability of Löhr's original formulation of DPC.

To do so we have extended the FSTRIPS [17] modeling language to enable the compact description of the plant, low–level control signals and modes without need of off–line pre–processing. FSTRIPS is a general language for classical planning based on the fragment of First Order Logic (FOL) that involves *constant*, *functional* and *relational* symbols (predicates), as originally proposed by Geffner, and recently augmented by Francès et al with support for quantification, conditional effects [15], and arbitrary procedural extensions to denote functions [16]. These features allow us to integrate in a parsimonious manner the syntactic fragments of PDDL 2.1 Level 2 [13] and PDDL+ [14] (PDDL 2.1 Level 5), necessary to account for numeric variables, general algebraic expressions, ordinary differential equations (ODE's), instantaneous *autonomous* transitions and support for the translation of existing models of hybrid systems. For each control cycle t a (ground) FSTRIPS planning problem $P_t = \langle V, D, s_0^t, A, C, \phi_G, q, p \rangle$ is set. V and D are state variables and their domains, $|V| = m + n$. States s_k are given as a valuation of state variables V, no actual difference is made between state and inputs. s_0^t is the initial state at cycle t. ϕ_G is the *goal formula* that specifies the properties that *goal states* must have. A and C are respectively a set of ground actions, and a set of *global constraints* (ground formulae) over V. Domains D and constraints C are used to construct automatically Equation 5, ϕ_G defines Equation 6. Our main extensions, beyond adding data types and algebraic functions, affect the way actions are defined and what is the interpretation made of their effects. A consists of three distinct subsets A_c, A_x and A_n corresponding to Reiters' *control*, *exogenous* and *natural* actions [42], that model respectively inputs u_k, instantaneous reactions of the environment and differential equations. p and q are the algebraic expressions over V in cost–to–go functions J like that in Eq. 8. We next discuss the semantics of our extensions.

[2] Control cycles are fixed duration time intervals, discretising the temporal evolution of the system. The number of control cycles in an ACE simulation is finite.

4.3 Actions with Procedural Effects

Geffner [17] defines effects of (ground) actions a as a set of conditional functional and relational effects e. Functional effects $\phi \rightarrow x := \omega$ *assign* to state variables x the value resulting of the evaluation of algebraic expression ω on a state s_k, noted $[\omega]^{s_k}$, whenever ϕ is true in s_k. Relational effects $\phi \rightarrow L$ where L is a conjunction of *literals* of state variables y whose domain D_y is $\{\top, \bot\}$. We will denote the state variable affected by effect e as aff_e. Relational effects are not strictly needed, yet making such distinction is convenient from a formal and practical aspect. We change Geffner's account in two ways. First, effects of actions are no longer *sets* but rather *sequences* $(e_1, \dots, e_k)$. The result of executing an action a becomes then the result of evaluating a simple *program* made up of k pairs of conditional assignment instructions, each of them addressed by the index $1 < j < k$ of the effect. This ensures that the result of evaluating the action effects is well–defined when $\text{aff}_{e_i} = \text{aff}_{e_j}$, for any two effects e_i, e_j, $i \neq j$. The second change we make to Geffner's account is to allow a third type of conditional effect, $\phi \rightarrow Y := \text{proc}(V)$, where Y is a tuple of state variables $Y \subseteq V$. $\text{proc}(V)$ is an *arbitrary* procedure whose interpretation $[\text{proc}(V)]^s$ depends on the values of V in state s and the programming language used[3].

4.4 Instantaneous Transitions

Controlled mode transitions are handled with *control* [42] actions A_c that the planner will be expected to make a decision on whether to execute or not. Allowing for procedural action effects greatly increases the expressiveness of the planning language, enabling for instance to incorporate domain control knowledge in a seamless and elegant way. For such actions, preconditions can be used to account for the guards governing the activation of action nodes in a behaviour tree, or state–based outputs in a general automaton. Effects are defined procedurally and set the values of either state or input variables. *Uncontrolled* mode transitions are interpreted as *exogenous* [42] actions A_x, which implement a special case of Fox & Long PDDL+ *events* [14]. We restrict the interpretation of exogenous action effects to sets where left and right hand sides are *independent*. The process of determining which actions are applicable on a given state and the execution of their effects is encapsulated in the procedural effect $propagate(V)$. While very limiting, this restriction is sufficient to meet the needs of the application we consider on this paper. Namely, they are used to account for *reactions* of the environment to inputs or the continuous evolution of dynamics. An example of such reactions are used to set the *commanded* bank angle and the error signal that governs bank angle rates $\dot{\phi}$. They can also be used to implement *supervisory control* [28] rules, which in this paper are used to model the pilot agent of adversaries.

4.5 Plant and Low-Level Control Dynamics

Plant and controller dynamics are conceptually *natural actions* [42] A_n, and equivalent to PDDL+ *processes* [14], which are used to describe the dynamics of the system in a compositional, modular, bottom up fashion. Preconditions of processes p, pre_p, can be used to account for the *guards* associated with the modes of the high–level controller and define the partitions $\mathcal{M}_k$, or when necessary, to approximate piecewise non–stationary plant dynamics. The effects eff_p of processes p are then combined to obtain the differential inclusions f_k directly, as eff_p are sets of ODE's

$$\text{eff}_p = \{\dot{x} = \xi \mid x \in V\}$$

where ξ is allowed to be an arithmetic expression involving standard arithmetic operations, any state variable $y \in V$, constants, built–in algebraic functions, or arbitrary functions whose codomain is defined by means of an external procedure [17]. The passage of time is modeled by adding automatically a process p_{clock} with effect $\dot{t} = \Delta t$ and precondition $\top$. Given a state s_k, the set of *active processes* [14] is defined as

$$\mathcal{P}_k = \{p \mid [\text{pre}_p]^{s_k} = \top\} \tag{10}$$

the set of processes whose preconditions are true. For each possible set $\mathcal{P}_k$ there is a mode with partition $\mathcal{M}_k$ given by the conjunction of preconditions pre_p, $p \in \mathcal{P}_k$ and ODE's for each variable $x \in V$

$$f_k^x = \left[\sum_{p \in \mathcal{P}_k} \text{eff}_p^x\right] \tag{11}$$

where eff_p^x is the expression that results from evaluating the right hand side of $\dot{x} = \xi$ on s_k, and x appears in the left–hand side, or 0 if there is no such ODE. We note that f_k^x is generally not constant throughout control cycles, as the terms featured in expressions ξ can have their denotation changed as the effect of instantaneous *control* or *exogenous* actions. The state transition function $f_i(\cdot)$ in Eq. 4 is then obtained solving *numerically* [6] the *integral equation*

$$x^{k+1} = x^k + \int_0^{\Delta t} f_k^x \, \partial\tau \tag{12}$$

for each variable $x \in V$, where x^k is the value of x in state s_k. As long as the Lifschitz condition holds over the interval $[0, \Delta t]$ for every f_k^x, Equation 12 will produce *finite* values. The computation of sets $\mathcal{P}_k$, differential equations f_k^x and the solution Eq. 12 is all done in our planner by the procedure $sim(V)$. It evaluates process preconditions and sets Equation 12 automatically. Once the state equation is established, $sim(V)$ proceeds to calculate the values x^{t+k+1} applying the integration method of choice, selected during the planner setup. We conclude this Section by observing that the use of numeric methods in hybrid planning is not entirely novel, both Dellapenna et al [11] UPMurphi, Piotrowski et al [40] DiNo and Scala et al [45] ENHSP planners do so implicitly relying on the Euler method [6].

4.6 Plans as Time–Switched Hybrid Systems

At each control cycle t, we invoke an off-the-shelf, deterministic planner on P_t. Resulting plans π_t are *finite sequences*

$$a_0^0, \dots, a_{l_1}^0, wait_1, a_0^1, \dots, a_{l_2}^1, wait_2 \dots a_{l_i}^{i-1}, wait_i, \dots, b$$

made of control actions $a_j^i \in A_c$, the special action *wait*, with procedural effect $Y := sim(propagate(V))$, ending with action $b \in A_c \cup \{wait\}$. By composing the procedures $propagate()$ and $sim()$ defined over the sets of exogenous and natural actions $A_x \cup A_n$ as discussed in Sections 4.4 and 4.5, *wait* effectively approximates via numeric integration the continuous time evolution of the control

[3] We use C++ in our planner.

system for over the period of time Δt. Plans π are *valid* when

$$\left[x_i, u_i\right]^T = \Lambda_i(x_{i-1}, u_{i-1}) \tag{13}$$

$$x_{i+1} = \text{eff}_{wait_i}(x_i, u_i) \tag{14}$$

$$h_C(x_i, u_i) \geq 0 \tag{15}$$

$$x_N \in \mathcal{X}_f \tag{16}$$

$$x_0 = x(t) \tag{17}$$

have at least one solution. Λ_i is the *numeric kernel* of action sequence $a_0^{i-1}, \ldots, a_{l_i}^{i-1}$ [44]. Namely, the *expression* that results from the recursive substitution of occurrences of variable y on the right hand side of the effect of action $a_{l_i}^i$ for the conditional effect

$$y := pre\left(a_{l_i}^k\right) \wedge \phi \rightarrow \omega$$

that results from effects $y := \phi \rightarrow \omega$ in $\text{eff}(a_{l_i}^k)$, with $k < i$ s.t. no action $a_{l_i}^m$, $k < m < i$ exists with an effect on y. If no action affects y, then $y_i = y_{i-1}$. We observe that Eqs. 13–17 describe all possible trajectories of a *time–switched* hybrid system [29], where the timing of the switches between modes is given by $t + i\Delta t$. Plans π_t are used directly to provide the output of the controller $g_{MPC}(x(t))$, set to be the state resulting from the execution of *instantaneous* actions $a_0^0, \ldots, a_{l_1}^0$.

4.7 Approximation of Optimal Control with Width-Based Search Methods

As we have seen above, FSTRIPS planning tasks P_t describe compactly deterministic optimal control problems over hybrid systems. In order to obtain the controller $g_{MPC}(x(t))$ efficiently, we turn to *Approximate Dynamic Programming* (ADP) [4]. In particular, we instance the *rollout* algorithm with a l–step, depth–selective *lookahead* policy. In contrast with well–known algorithms such as Monte Carlo Tree Search (MCTS) [27], our lookahead and base policies are deterministic.

Rather than using a simulation-based policy to estimate the cost $\tilde{J}_{k+l}$ at the leaf nodes of the lookahead, as MCTS does, we simply set $\tilde{J}_{k+l}$ to $g_N(x_{k+l})$. While this may seem to be a rather unsophisticated choice, we have observed it to perform very robustly. For the lookahead policy we turn to *Width-based Search* [30]. These algorithms both allow to focus the lookahead and have good any–time behaviour. When it comes to prioritisation of applicable control actions, width–based methods select first those that lead to states with novel valuations of features defined over states [16]. For the experiments in Section 6, we have implemented the simplest width–based algorithm, $IW(1)$, to unroll the lookahead for l steps. Similar strategies have been shown to perform well over fixed horizon deterministic and stochastic control problems [18, 31]. $IW(1)$ is a plain *breadth–first search*, guaranteed to run in *linear time and space* as it only expands *novel* states. A state $s_k = [x_k\ u_k]^T$ is *novel* if and only if it encounters a state variable x or u whose value $D(x), D(u) \subseteq \mathbb{R}$ it has not seen before in the current search. Note that novel states are independent of the objective function used, as the estimated cost-to-go J is not used to define the novelty of the states. Hence, novelty–based lookahead policies are orthogonal to other strategies, such as *regret* minimization [27] and the evaluation of greedy policies over approximations of cost–to–go functions, as used by Real–Time Dynamic Programming (RTDP) [2] and Deep Q-Learning [36]. How to combine existing cost–based approaches with width–based ones is an active topic of research.

5 FSTRIPS MODEL OF PLANT AND CONTROL DYNAMICS

We have modeled directly, without further simplifications, the plant dynamics described in Section 3.1. In order to reduce the error during numeric integration, we have grouped the ODE's in Eq. 1 into a single process. Rates of change for Euler angles and speed are modeled independently, and depend on the control mode active at a given time point. The FS planner analyses the input and output dependencies between the integration actor blocks [28] and performs leapfrogging automatically if no cyclic dependencies are found.

We have used a different control scheme than the one described in Section 3.1, following the guidelines for the design of supervisory control systems provided in Lee & Seeshia textbook [28]. Rather than trying to find values for the commands to be used in the flight control system, we introduced 3 control modes for each of the 3 degrees of freedom: pitch, bank angle and speed. Each of these *primitive* control modes specify whether either of these quantities are to decrease (increasie) according to a simple linear law, or remain stationary, during the control cycle. The control mode $(\mathcal{M}_{t+k}, f_{t+k})$ follows from the combination of valid values of 3 different multi–valued logical variables, each one with a domain consisting of 3 values, one for each of the primitive control modes considered. Each of these can be activated via suitable control actions *accelerate*, *decelerate*, *cruise*, etc. whose effects set the corresponding logical variable. The *wait* action discussed in Section 4.6, will then resolve which of the possible $3^3 = 27$ permutations of primitive control modes needs to be used to compute the next state. Orthogonal to these differential, proportional control modes, we allow the planner to determine dynamically the maximum g-load [25] allowed during a given control cycle. For that, we have included 3 control actions, *increase_g*, *decrease_g* and *reset_g* that increase (decrease) by 1 unit the current g-load of the aircraft and reset it to 1. Besides that, these actions also set the commanded bank angle ϕ_c of the aircraft according to the formula $\phi_c = \text{acos}\, g_{load_c}^{-1}$. We note that an aircraft with a g-load of 1 will not start turning. If it is already turning the bank angle will eventually become zero, so the aircraft flies again along a straight line.

Obtaining command values for the control scheme used by ACE simulated aircraft is straightforward, and results from the numerical simulation of the time–switched hybrid system given by the plan π_t computed at control cycle t over the horizon N. We do not use directly the values for ψ and θ at terminal states as they can have very small magnitudes, and we just keep their sign which we multiply by constants $\psi_t = \pi/2$ and $\theta_t = 0.017rad$.

6 EXPERIMENTAL EVALUATION

We end this paper discussing the parameters under which we have studied the performance of the MPC controllers based on planners discussed in Section 4. We first define formally the goal and cost function used in our experiments. Then we give a detailed account of the main parameters governing the simulation runs. Finally, we go over the observations obtained on the performance of our controllers compared with the baseline described in Section 3.2.

6.1 Goal & Cost Function Formulation

Goal states s_G are those where the goal formula $\phi_G = R_{min} \leq R_{br} \leq R_{max} \wedge 0 \leq AA_b \leq 60 \wedge 0 \leq ATA_b \leq 30$ holds. R_{min} and R_{max} are the lower and upper bounds on the range to the target and are set to 100 and 1, 000 respectively. AA and ATA are, respectively the *aspect* and *antenna train* angles. They describe in a compact manner the tactical relations between the two UAVs [1, 35].

The cost function J we use in these experiments is task specific, and originally developed by McGrew et al [35]. McGrew's function is built around an expert developed heuristic [1], that captures the merit of states s_k and considers relative aircraft orientation and range

$$h(s_k) = \left(\frac{1}{2} - \left(\frac{AA}{2\pi} + \frac{ATA}{2\pi}\right)\right) \exp\left(\frac{-|R - R_d|}{\pi\lambda}\right) \tag{18}$$

where AA is the aspect angle to the target aircraft, ATA is the antenna train angle, R is the distance between Blue and Red, R_d is the *desired* range to be maintained and λ is a scaling constant. We set $\lambda = 1,000$ to keep it in line within the same order of magnitude of initial distances between Red and Blue. Equation 18 combines two measures of performance. The first term, which is a function of the aspect and antenna train angles is referred by McGrew as the *orientation score*, and rates highest states where the controlled UAV is right behind the target. These values lie in the $[-1, 1]$ interval, and they are symmetric for Blue and Red. That is, when Blue orientation score is 1, Red's assigned score is -1. If Blue and Red are flying side by side, then scores are 0 for both. The second term, the *range score*, is a function mapping distance to the target into the $[0, 1]$ interval, assigning 1 when the range to the target is exactly at a distance R_d, and gently falling off according to an inverse exponential law. The stage cost $q(x_k, u_k)$ combines h

$$q_{sc}(x_k, u_k) = w\, g(x_k) + (1 - w)h(x_k) \tag{19}$$

with the *goal indicator function* $g(x_k)$, which is 1 whenever the controlled UAV is right within the goal region in Equation 18, and 0 otherwise. The constant $w \in [0, 1]$ is a weighting value determined experimentally, and we use $w = 0.8$ as suggested by McGrew in [35]. We have found experimentally that McGrew's approach does not suit the larger distances involved in our setting, and rather than multiplying the orientation and range scores, we have found that adding them together produces best results with the algorithms discussed in Section 4.7 confirming the observations in [39]. We finish noting that McGrew's terminal cost $p(\cdot)$ is set to zero, and we do so too.

6.2 Planner-based Controllers

We have tested 9 different planning–based controllers, all of them using the FS planner [16]. The planner has been extended to support the width–based search algorithms by Lipovetzky et al [31] tested on the ATARI simulator, and the new FSTRIPS features discussed in Section 4. Each of these controllers follow from the discussion in Section 5, with some changes which we discuss next. We have considered three different supervisory control schemes. The first one, which we refer to as *Vanilla*, corresponds exactly with the one presented in Section 5. The second scheme tested, which we will refer to as *Composite* and abbreviate as *Comp.*, includes actions with procedural effects that encapsulate the action nodes of the behaviour tree in Figure 3. Besides those, other control actions available are those that switch the system back to neutral control modes (e.g. *cruise*) and allow to adjust g_{load_c}. This allows the planner to explore more aggressive turning and interleave the pursuit of the sub-goals discussed in Section 2 in novel ways. The third variant considered, which we call *Shaw*, control actions allow to adjust g_{load_c} and there is one action whose procedural effect evaluates the *complete* behaviour tree in Figure 3. The second dimension in the controllers incorporates models of the opponent by assuming the target aircraft to be governed by a specific supervisory control strategy. The first assumed control scheme, *Straight*, assumes the target to fly in the same direction and velocity. The second opponent model considered evaluates and executes the *Pure Pursuit* behaviour in Figure 3. The third and last model considered, *Full Shaw*, evaluates and executes the complete behaviour tree in the same Figure. The two are implemented via exogenous actions that update opponent control inputs each control cycle.

6.3 Experimental Settings

While initial conditions (positions, velocities) of Blue and Red change between simulations a number of parameters and settings remain constant. We discuss these invariant properties next. To simplify the analysis and avoid having to adjust the parameters in the behaviour tree in Figure 3 we have used the same airframe model in all simulations. This fixes the values for the *maximal turn rate* (set to 4 degrees per second), *thrust-to-mass* ratio (yielding a maximum linear acceleration of 5 m/s^2) and the critical *stall* speed (set to 80 m/s). The duration of simulations is set to be 600 seconds, and the frequency of updates and control cycles is set to 10 Hz, so every history generated by the simulation consists of 6, 000 cycles. We rate the performance of Blue and Red using the *orientation score* defined in Section 6.1. Since it is symmetric, the sum of attained scores through the simulation history is both an intuitive and meaningful measure of relative performance. When the accumulated score is close to zero, it follows that either Blue and Red flew along parallel course most of the time, or more interestingly, positional advantage changed in a balanced way through the simulation. The depth of the lookahead l is set to 1 second, e.g. 10 control cycles, in all the experiments. This allowed to evaluate up to ≈5,000 states[4] ahead from the current state $x(t)$ on average, keeping the run-time associated to each call to the planner to be about 50 ms, well below the 100 ms budget allowed by the simulation time step.

6.4 Summary of Results

We have compared the 9 controllers discussed in Section 6.2 against the baseline pilot agent over a diverse array of initial conditions. These are classified as Neut*ral*, Off*ensive* and Def*ensive*, depending on the value of McGrew's orientation score in the first frame of the simulation history. Neutral initial conditions include situations where aircraft are approaching each other head on, a configuration

[4] The memory footprint of the search tree is ≈5 MBytes of memory.

Table 1: Evaluation of the 9 controllers proposed in Section 6.2 against the baseline discussed in Section 3.2.

Config.	Opp. ModelI		N	SM	t	p	CI_{lb}	CI_{ub}
★Vanilla	Straight	Def	32	750.180	5.176	1.30×10^{-6}	454.580	1045.780
Vanilla	Straight	Neut	229	927.185	11.268	1.06×10^{-23}	765.044	1089.326
Vanilla	Straight	Off	39	883.935	5.674	1.59×10^{-6}	568.566	1199.303
Vanilla	Pursuit	Def	32	648.529	5.136	1.45×10^{-5}	391.000	906.058
★ Vanilla	Pursuit	Neut	229	929.402	10.240	1.73×10^{-20}	750.558	1108.246
Vanilla	Pursuit	Off	39	1182.616	6.502	1.17×10^{-7}	814.411	1550.821
Vanilla	Full Shaw	Def	32	647.944	5.131	1.48×10^{-5}	390.386	905.502
Vanilla	Full Shaw	Neut	229	859.442	9.632	1.22×10^{-18}	683.624	1035.261
★Vanilla	Full Shaw	Off	39	1183.473	6.514	1.13×10^{-7}	815.663	1551.283
Comp.	Straight	Def	32	-667.954	-2.568	0.015	-1198.368	-137.539
Comp.	Straight	Neut	229	-631.595	-7.965	7.83×10^{-20}	-787.849	-475.340
Comp.	Straight	Off	39	107.177	0.503	0.618	-324.028	538.383
Comp.	Pursuit	Def	32	-747.461	-2.947	0.006	-1264.704	-230.219
Comp.	Pursuit	Neut	229	-705.692	-8.389	5.15×10^{-15}	-871.444	-539.940
Comp.	Pursuit	Off	39	-545.544	-2.355	0.024	-1014.462	-76.626
Comp.	Full Shaw	Def	32	-775.177	-3.042	0.005	-1294.913	-255.441
Comp.	Full Shaw	Neut	229	-709.601	-8.428	4.01×10^{-15}	-875.509	-543.692
Comp.	Full Shaw	Off	39	-538.705	-2.327	0.025	-1007.296	-70.114

typically considered in the literature to be “fair” as it’s not offering undue advantage to either aircraft [35, 39]. Defensive initial conditions are those where the target aircraft starts *behind*, and conversely, offensive initial conditions are those where the agent starts behind the target already.

Table 2: Probability of success planning–based controllers and the Shaw heuristic when initial conditions are neutral.

Configuration	Opponent Model	Prob. Alone	Prob. Both
Vanilla	Straight	0.175	0.192
Vanilla	Pursuit	0.166	0.157
Vanilla	Full Shaw	0.170	0.157
Composite	Straight	0.214	0.087
Composite	Pursuit	0.197	0.122
Composite	Full Shaw	0.197	0.122
Shaw’s Heuristic	N/A	0.035	0.062

Table 1 shows a statistical analysis of the performance observed of each planner configuration. For each combination of planner configuration (Vanilla, Composite, Shaw) and opponent modeling (Straight Flight, Pursuit, Full Shaw) we measure the mean orientation score as defined in equation 18 (column *SM*) throughout N simulations for each of the three classes of initial conditions discussed above. To assess the significance of this observable, we have conducted a two–sided t-test for it (column t is the statistic, column p is the p-value) and calculated the 95% confidence interval values (column *CI*). The null hypothesis tested is that the population or *actual* mean accumulated score is 0. In other words, we check to what degree the experimental observations support the hypothesis that planners and Shaw Heuristic performance is so close that their scores are both very close to zero. Our interpretation of the results and statistics reported on Table 1 is that configurations using the Vanilla control scheme have generally attained mean scores statistically significantly superior to those of Shaw’s heuristic. The very low p-values for the null hypothesis and sign of the t statistic suggest that indeed, it is very likely that these controllers are slightly superior, over all initial conditions, to our baseline. We can then conclude that the FS planner, operating over the domain theory described in Section 5, with and without modeling of opponent actions, have a performance superior, as measured by McGrew’s scoring functions, to Shaw’s proposed technique. We note that this outcome is obtained directly from the *first principles* that follow from the dynamical constraints in the flight model and the structure of the cost function defined in Section 6.1. The Composite and Shaw variants were found to be on par with Shaw’s heuristic in situations where the aircraft controlled by the planner starts with a tactical advantage, we omit to report these results due to lack of space. Our statistical significance test rules out the null hypothesis in Neutral situations. Our interpretation is that the planner does not find useful variants on Shaw’s rules of thumb for those particular configurations.

Table 2 reports the success probabilities for the planner and the implementation of Shaw’s heuristic in Figure 3. The column “Prob. Alone” in the Table reports the relative frequency of simulations where the pilot agent corresponding to each row achieved the task, satisfying ϕ_G for 50 *consecutive* control cycles, *and* also managed to avoid its opponent to do the same. The column “Prob. Both” reports the frequency in which agents achieved the goal yet could not avoid the other aircraft from doing so too. Interestingly, this Table shows a picture which is complementary to Table 1. With this measure of performance, we see that the 9 controllers are far more likely to achieve the goal than Shaw’s heuristic. It is remarkable that the Composite and Shaw controllers, which are outperformed in Table 1, are on the other hand quite effective to achieve the goal. These results hold witness to the highly dynamic nature of the task, as the tables can be turned on the opponent several times over the duration of each simulation.

7 DISCUSSION & FUTURE WORK

The results presented in the previous Section demonstrate that approximate dynamic programming techniques based on Lipovetzky & Geffner width–based search framework, running on top of general simulators described symbolically via FSTRIPS, are a viable approach to guidance and control in settings that preclude direct application of MILP–based approaches [5, 43]. Previously reported results [41] showed the approach to be superior in run–time and performance to game–theoretic heuristic methods [39]. We look forward to compare directly with related dynamic programming approaches such as Monte Carlo Tree Search [27] and Deep Reinforcement Learning [36, 47] methods, over several domains, considering stochastic perturbation and partial observability. Furthermore we seek to reduce the amount of domain expertise required by the approach, tapping into the power of state–of–the–art machine learning [21] to acquire models of systems and discover modes of low–level control signals relevant to the task [37] directly from simulators.

ACKNOWLEDGMENTS

This work has been partially funded by the Australian Defence Science Institute sponsored research grant program CERA 2017.

REFERENCES

[1] Fred Austin, Giro Carbone, Michael Falco, and Hans Hinz. 1990. Game Theory for Automated Maneuvering During Air-to-Air Combat. *Journal of Guidance, Control and Dynamics* 13, 6 (1990), 1143–1149.

[2] Andy G. Barto, S. J. Bradtke, and S. P. Singh. 1995. Real–Time Learning and Control Using Asynchronous Dynamic Programming. *Artificial Intelligence Journal* 72 (1995), 81–138.

[3] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. 2002. The explicit linear quadratic regulator for constrained systems. *Automatica* 38, 1 (2002), 3–20.

[4] Dmitri P. Bertsekas. 2017. *Dynamic Programming and Optimal Control* (4th ed.). Athena Scientific.

[5] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. 2017. *Predictive control for linear and hybrid systems.* Cambridge University Press.

[6] John C. Butcher. 2008. *Numerical Methods or Ordinary Differential Equations* (2nd ed.). Wiley & Sons.

[7] Michael W. Byrnes. 2014. Nightfall: Machine Autonomy in Air-to-Air Combat. *Air and Space Power Journal* (May-June 2014).

[8] Stefano Di Cairano. 2012. An industry perspective on MPC in large volumes applications: Potential Benefits and Open Challenges. *IFAC Proceedings Volumes* 45, 17 (2012), 52–59.

[9] Eduardo F. Camacho and Carlos Bordons. 2013. *Model predictive control* (3rd ed.). Springer Science & Business Media.

[10] Michele Colledanchise and Petter Ögren. 2017. How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees. *IEEE Transactions on Robotics* 33, 2 (April 2017), 372–389. https://doi.org/10.1109/TRO.2016.2633567

[11] Giuseppe DellaPenna, Daniele Magazzeni, Fabio Mercorio, and Benedetto Intrigila. 2009. UPMurphi: a tool for universal planning on PDDL+ problems. In *Proc. of the Int'l Conf. in Automated Planning and Scheduling (ICAPS).*

[12] Rick Evertsz, John Thangarajah, Nitin Yadav, and Thanh Ly. 2015. A Framework for Modelling Tactical Decision-making in Autonomous Systems. *J. Syst. Softw.* 110, C (Dec. 2015), 222–238. https://doi.org/10.1016/j.jss.2015.08.046

[13] Maria Fox and Derek Long. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20 (2003), 61–124.

[14] Maria Fox and Derek Long. 2006. Modelling Mixed Discrete-Continous Domains for Planning. *Journal of Artificial Intelligence Research* 27 (2006), 235–297.

[15] Guillem Frances and Hector Geffner. 2016. ∃-STRIPS: Existential Quantification in Planning and Constraint Satisfaction. In *Proc. of Int'l Joint Conf. in Artificial Intelligence (IJCAI).*

[16] Guillem Francès, Miquel Ramirez, Nir Lipovetzky, and Hector Geffner. 2017. Purely Declarative Action Descriptions are Overrated: Classical Planning with Simulators. In *Proc. of Int'l Joint Conf. in Artificial Intelligence (IJCAI).*

[17] Héctor Geffner. 2000. Functional STRIPS: a more flexible language for planning and problem solving. In *Logic-based artificial intelligence*, Jack Minker (Ed.). Springer, 187–209.

[18] Tomás Geffner and Hector Geffner. 2015. Width-based Planning for General Video-Game Playing. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.*

[19] Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: theory and practice.* Elsevier.

[20] Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. 2009. Hybrid Dynamical Systems. *IEEE Control Systems Magazine* 29, 2 (2009), 28–93.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* The MIT Press.

[22] Clinton Heinze, Michael Papasimeon, and Simon Goss. 2000. *Specifying Agent Behaviour with Use Cases.* Springer Berlin Heidelberg, Berlin, Heidelberg, 128–142. https://doi.org/10.1007/3-540-44594-3_10

[23] Clint Heinze, Michael Papasimeon, Simon Goss, Martin Cross, and Russell Connell. 2008. *Simulating Fighter Pilots.* Birkhäuser Basel, Basel, 113–130. https://doi.org/10.1007/978-3-7643-8571-2_7

[24] Clint Heinze, Bradley Smith, and Martin Cross. 1998. *Thinking quickly: Agents for modeling air warfare.* Springer Berlin Heidelberg, Berlin, Heidelberg, 47–58. https://doi.org/10.1007/BFb0095040

[25] David G. Hull. 2007. *Fundamentals of Airplane Flight Mechanics* (2nd ed.). Springer.

[26] Falilat Jimoh and Thomas Leo McCluskey. 2016. Towards The Integration of Model Predictive Control into an AI Planning Framework. In *UK Planning Special Interest Group Workshop.*

[27] Levente Kocsis and Csaba Szepevari. 2006. Bandit Based Monte Carlo Planning. In *Proc. of European Conference in Machine Learning (ECML).*

[28] Edward A. Lee and Sanjit A. Seshia. 2016. *Introduction to embedded systems: A cyber-physical systems approach.* MIT Press.

[29] Daniel Liberzon. 2012. *Switching in Systems and Control.* Springer Science & Business Media.

[30] Nir Lipovetzky and Héctor Geffner. 2012. Width and Serialization of Classical Planning Problems. In *Proc. of European Conference on Artificial Intelligence (ECAI).*

[31] Nir Lipovetzky, Miquel Ramirez, and Hector Geffner. 2015. Classical planning with simulators: results on the Atari video games. In *Proc. of Int'l Joint Conf. in Artificial Intelligence (IJCAI).*

[32] Johannes Löhr, Patrick Eyerich, Thomas Keller, and Bernhard Nebel. 2012. A Planning Based Framework for Controlling Hybrid Systems.. In *Proc. of the Int'l Conf. in Automated Planning and Scheduling (ICAPS).*

[33] Kevin McDonald, Lyndon Benke, and Michael Papasimeon. 2015. Team Oriented Execution Models for Multi-Agent Simulation of Air Combat. In *Proceedings of the 21st International Congress on Modelling and Simulation (MODSIM 2015).*

[34] Kevin McDonald and Michael Papasimeon. 2015. Augmented Reality as an Interface to Air Combat Multi-Agent Simulation. In *Proceedings of the 2015 Simulation and Technology Conference (SimTect 2015).*

[35] James S. McGrew and Jonathan P. How. 2010. Air Combat Strategy using Approximate Dynamic Programming. *Journal of Guidance, Control and Dynamics* 33 (2010), 1641 – 1654.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (2015), 529.

[37] Igor Mordatch, Nikhil Mishra, Clemens Eppner, and Pieter Abbeel. 2016. Combining model-based policy search with online model learning for control of physical humanoids. In *Proc. of the IEEE Int'l Conference on Robotics and Automation (ICRA).*

[38] Michael Papasimeon, Lyndon Benke, Richard Brain, and Lily Finkelshtein. 2018. Multiagent Simulation of Adversarial Socio-Technical Systems. In *Proc. of the Int'l Conference on Autonomous Agents and Multiagent Systems.*

[39] Hyunju Park, Byung-Yoon Lee, Min-Jea Tahk, and Dong-Wan Yoo. 2016. Differential Game Based Air Combat Maneuver Generation Using Scoring Function Matrix. *International Journal Of Aeronautical AND Space Sciences* 17, 2 (2016), 204–213.

[40] Wiktor Mateusz Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio. 2016. Heuristic Planning for PDDL+ Domains. In *Proc. of Int'l Joint Conf. in Artificial Intelligence (IJCAI).*

[41] Miquel Ramirez, Michael Papasimeon, Lyndon Behnke, Nir Lipovetzky, Tim Miller, and Adrian R. Pearce. 2017. Real–Time UAV Maneuvering via Automated Planning in Simulations. In *Proc. of Int'l Joint Conf. in Artificial Intelligence (IJCAI).*

[42] Raymond Reiter. 2001. *Knowledge in action: logical foundations for specifying and implementing dynamical systems.* MIT press.

[43] Arthur Richards and Jonathan P. How. 2003. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *American Control Conference.*

[44] Enrico Scala. 2013. Numeric Kernel for Reasoning about Plans Involving Numeric Fluents. In *AI*IA 2013: Advances in Artificial Intelligence: Proc. Int'l Conf. of the Italian Association for Artificial Intelligence*, Matteo Baldoni et al (Ed.). Springer International Publishing, 263–275.

[45] Enrico Scala, Patrik Haslum, Sylvie Thiebaux, and Miquel Ramirez. 2016. Interval–Based Relaxation for General Numeric Planning. In *Proc. of European Conference on Artificial Intelligence (ECAI).*

[46] Robert L. Shaw. 1985. *Fighter Combat: Tactics and Maneuvering.* Naval Institute Press.

[47] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529 (2016), 484–489.

[48] Gil Tidhar, Clinton Heinze, and Mario Selvestrel. 1998. Flying Together: Modelling Air Mission Teams. *Applied Intelligence* 8, 3 (1998), 195–218. https://doi.org/10.1023/A:1008271016283