Dynamic Chess: Strategic Planning for Robot Motion

Tobias Kunz, Peter Kingston, Mike Stilman and Magnus Egerstedt

***Abstract*— We introduce and experimentally validate a novel algorithmic model for physical human-robot interaction with hybrid dynamics. Our computational solutions are complementary to passive and compliant hardware. We focus on the case where human motion can be predicted. In these cases, the robot can select optimal motions in response to human actions and maximize safety. By representing the domain as a Markov Game, we enable the robot to not only react to the human but also to construct an infinite horizon optimal policy of actions and responses. Experimentally, we apply our model to simulated robot sword defense. Our approach enables a simulated 7-DOF robot arm to block known attacks in any sequence. We generate optimized blocks and apply game theoretic tools to choose the best action for the defender in the presence of an intelligent adversary.**

I. Introduction

In order to deploy safe and flexible robots for service and automation, robots must act safely in close contact with humans. This includes positive interactions such as collaborative tasks and the absence of negative interactions such as human-robot collisions. The goal of this work is to optimize robot motion such that it maintains maximal safety regardless of any change in the environment including the most dangerous choices made by humans who share the robot's workspace.

In order to maximally protect humans and robots, the robot should take into account all possible situations and plan for the worst. Such reasoning is similar to decision making in strategy games like chess where the players not only choose good moves for themselves but also moves that prevent their opponent from winning. The robot wins when safety is maintained. It loses when the opponent manages to instigate danger. In contrast to chess, which is entirely discrete, the dynamics of motion have both discrete and continuous components: Human choices of action types may be discrete, yet the actual motion involved in executing each action is continuous. Likewise, the safest robot response also has both discrete and continuous components. We introduce a theoretical model that represents hybrid physical games and present a method for solving these games in real-time.

Our approach is validated in the domain of human-robot fencing. This application represents some of the most critical challenges for dynamic interactions between robots and humans: (1) prediction of human intentions and (2) real-time robot response. For this example, we concentrate on the following task: A robot arm, equipped with a sword must defend from a series of attacks initiated by a similarly-armed human adversary (Fig. 1).

The authors are with the Center for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA, USA tobias@gatech.edu, kingston@gatech.edu, mstilman@cc.gatech.edu, magnus@ece.gatech.edu

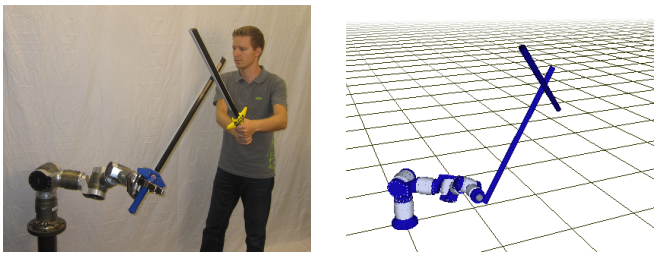


Fig. 1: ConAn fencing demonstrates discrete strategies in continuous domains.

II. Related Work

Prediction of human intentions by robots has been studied in both collaborative and antagonistic settings. In the collaborative context, both human-in-the-loop control methodologies (e.g. [1], in which an automatic controller works in concert with a human to achieve an objective), and improved human-robot interfaces (as investigated in [2] and [3], and motivated by field studies like [4]) may go a long way to achieving this. In short, humans should be able to communicate their intentions to a robot directly, efficiently trading the amount of direction required from the human, and the amount of estimation required of the robot.

In the antagonistic setting, humans may attempt to actively *hide* their intentions from the robot. This idea is explored explicitly in a somewhat different context in [5]), which compared to the collaborative setting raises both new challenges and new problem structure that can be utilized. On the one hand, the estimation task is complicated by the possibility of deception. On the other there exists game structure, known to both the human and the robot, which can be used inform the robot's decisions. We take advantage of game structure.

With regard to estimation, ongoing work in numerous fields classifies and predicts human behavior. For instance, there exists considerable work on *plan recognition* [6], [7]. More recently, in [8], the authors develop a hybrid Hidden Markov Observer to estimate human intent for active prosthetic and cooperative assembly applications. From a slightly different angle, [9] uses Dynamic Bayesian Networks to attach intent labels to observed actions.

Other work investigates adversarial interactions from a game-theoretic perspective. Much of this domain, motivated by robotic soccer competitions like RoboCup, considers robots that play against other robots. For instance, in [10], the authors advance zero-sum Markov Games as a natural framework for handling adversarial scenarios. They describe a modified Q-learning algorithm that learns optimal stationary policies that play these games. One application is a simplified model of robot soccer. Closely-related work includes [11] which gives a dynamic programming algorithm for solving partially-observable stochastic Markov Games,

and [12], which considers methods for approximate dynamic programming. Our approach is strongly influenced by literature on Markov Games. We contribute human-robot interaction and formalize the relationship between discrete and continuous components of moves.

III. Hybrid Adversarial Interaction Model

In the subsections that follow, we describe a hybrid-time model for adversarial interaction between two players. The first, dubbed the *attacker*, has the initiative, and is tasked with performing actions, called attacks, that oppose the interests of the second player, who we call the *defender*. The defender seeks to avoid this by reacting to the attacker's actions in a way that simultaneously prevents the attacker from succeeding in the present, and that also increases future attack difficulty.

A. Continuous Dynamics and Discrete Moves

The interaction model under consideration is formed from the interplay of the continuous and the discrete. The two players are assumed to control real physical systems, and their movements are both subject to differential constraints. Hence, the problem is continuous. However, both players are also assumed to have finite repertoires of behaviors. Thus the problem is also discrete. We start by defining the attacker's continuous-time dynamics as follows:

$$\dot{x}(t) = f(x(t), u(t)). \tag{1}$$

$x(t) \in \mathbb{R}^{n_x}$ is a real state vector and $u(t) \in \mathbb{R}^{m_x}$ is a control input that the attacker may vary. We assume that the attacker has a finite set of control input signals $u_1, \cdots, u_N$, called *attack inputs*, with $u_i : [0, \infty) \to \mathbb{R}^{m_x}$ for each $i \in \{1, \cdots, N\}$. Each attack input u_i is associated with an initial state $x_i^0 \in \mathbb{R}^{n_x}$, called the *precondition*, which the attacker must be in to begin the corresponding attack. Given the dynamics (1), each pair (u_i, x_i^0) induces a state trajectory $x_i : [0, \infty) \to \mathbb{R}^{n_x}$ called an *attack*. Finally, for each attack precondition and state $x^0 \in \mathbb{R}^{n_x}$, we assume the existence of a controller $k_i(x^0) : [0, T_i] \to \mathbb{R}^{m_x}$ such that if $x(0) = x^0$ and $u(t) = k_i(x^0)(t)$ then the dynamics (1) evolve such that $x(T_i) = x_i^0$. In other words, each precondition is reachable from anywhere in the state space, and for each precondition there exists a rule that drives the system there in finite time. We call this the *transition motion*.

The defender also has continuous-time dynamics:

$$\dot{y}(t) = g(y(t), v(t)), \tag{2}$$

with state $y(t) \in \mathbb{R}^{n_y}$ and control input $v(t) \in \mathbb{R}^{m_y}$. The discrete nature of the defender comes from a finite repertoire of *response functions* $r_1, \cdots, r_M$, with each $r_j : ([0, \infty) \to \mathbb{R}^{n_x}) \to \mathbb{R}_+ \times \mathbb{R}^{n_y}$, $r_j = (r_j^y, r_j^T)$ mapping from an observed attacker state trajectory $x_i \in ([0, \infty) \to \mathbb{R}^{n_x})$ to a desired point in time $r_j^T(x_i)$ within the attack trajectory, at which the attack is countered, together with a desired state $r_j^y(x_i) \in \mathbb{R}^{n_y}$ which the defender may reach in order to counter the attack. We assume that the defender uses a controller $\kappa : \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \times \mathbb{R}_+ \to \{v \mid v : [0, T] \to \mathbb{R}^{m_y}, T \in \mathbb{R}^+\} \cup \{\mathsf{X}\}$ that for each $T \in \mathbb{R}_+$ and points $y^0, y^1 \in \mathbb{R}^{n_y}$ either returns (1) a control input defined for $t \in [0, T]$ such that if $y(0) = y^0$ and $v(t) = \kappa(y^0, y^1, T)(t)$ in the dynamics (2), then $y(T) = y^1$; or (2) the null-move symbol "X," which indicates that no such control input exists.

```
HYBRID_DYNAMICS()
 1  for l ← 1 to ∞
 2  do i ← ATTACKER_PICKS_ATTACK
 3     j ← DEFENDER_PICKS_BLOCK(i)
 4     t_l ← t
 5     while t < t_l + T_i + r_j^T(x_i)
 6     do if t < t_l + T_i
 7           then ẋ(t) = f(x(t), k_i(x(T))(t − t_l))
 8           else ẋ(t) = f(x(t), u_i(t − t_l − T_i))
 9        ẏ(t) = g(y(t), κ(y(t_l), r_j^y(x_i)(t−t_l), T_i+r_j^T(x_i)))
```

Alg. 1: Hybrid dynamics

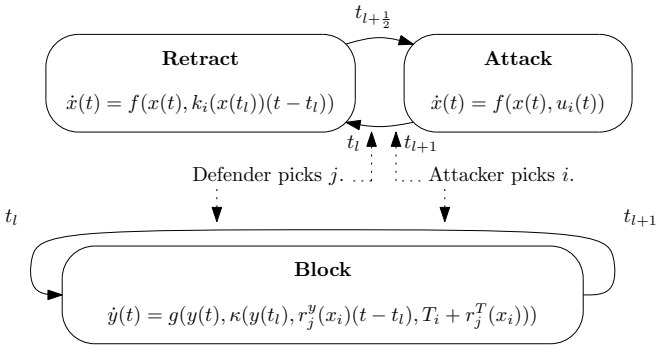


Fig. 2: Attacker and defender hybrid automata.

The attacker and defender then exchange moves in a sequence of attack-defense turns. The hybrid dynamics proceed as described by Fig. 2 and Alg. 1.

B. The Discrete State Space

The finite sets of attacks and response functions give rise to a finite state space. First, at any time $t \in \mathbb{R}$ we define the joint state for the two agents as

$$z(t) = (x(t), y(t)) \ . \tag{3}$$

Next, for each $(i, j) \in \{1, \cdots, N\} \times \{1, \cdots, M\}$, we define $z_{ij} \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_y}$ by

$$z_{ij} \triangleq (x_i(r_j^T(x_i)), r_j^y(x_i)). \tag{4}$$

Notice that $z(T) \in \{z_{11}, z_{12}, \cdots, z_{NM}\}$ is a loop invariant of algorithm 1. Hence, we define $\mathcal{Z} = \{z_{11}, z_{12}, \cdots, z_{NM}\} \sim \{1, \cdots, N\} \times \{1, \cdots, M\}$ to be the *discrete state space* of the given hybrid dynamics.

This state space can be visualized as a square $N \times M$ grid, similar to a chessboard. Suppose there exists a rook, which represents the current state. A turn of game play proceeds as follows. First, the attacker chooses a target row for the rook. Then, knowing this choice, the defender chooses a column. At this point the rook is moved to the square corresponding to the joint choice of row and column, and the next turn begins. This way the discrete state evolves into the hybrid

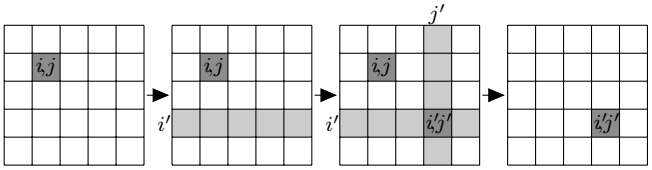


Fig. 3: A discrete two player game without continuous dynamics.

dynamics. Certain moves will turn out to be more difficult than others for the players to make (following a cost structure introduced in section III-D), and it will be each player's goal to force his opponent to make costly moves.

C. A Markov Game

A *Markov game* between K agents consists of a set $\mathcal{X}$ called the *state space*, together with a collection of action sets $\mathcal{A}_1, \cdots, \mathcal{A}_K$ (one for each agent), a set of cost functions $J_1, \cdots, J_K : \mathcal{Z} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_K \to \mathbb{R}$ that describe how undesirable any combination of actions in a state is for each agent, and a transition probability map $\rho : \mathcal{X} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_K \to \Delta(\mathcal{X})$ (where $\Delta(\mathcal{X})$ denotes the set of probability distributions over $\mathcal{X}$) that returns, for each state and any choice of actions by the players, the probability of transitioning to any other state. In other words, a finite Markov game is a controlled Markov chain [or Markov Decision Process (MDP)], where the control input at each step is determined not by a single decision-maker, but rather as a function of decisions made by several agents.

A Markov game is *zero-sum* if $J_1 + \cdots + J_K \equiv 0$ – i.e., if no matter what state the game is in or what actions the players take, the total cost is constant. Games of this type describe situations in which any player's gain necessarily comes at the expense of other players. It is *finite* if the state space $\mathcal{X}$ and action sets $\mathcal{A}_1, \cdots, \mathcal{A}_K$ are finite, *deterministic* if ρ returns a deterministic distribution (in which case, with some abuse of notation, we will write as though ρ maps directly to states; i.e., that $\rho : \mathcal{X} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_K \to \mathcal{X}$) for all inputs, and it is a *turn-taking game* if, in each state x, the map $(a_1, \cdots, a_K) \mapsto \rho(x, a_1, \cdots, a_K)$ depends on at most one of the a_is, $i \in \{1, \cdots, K\}$.

With these definitions, the adversarial interaction we have described in the previous sections can be posed as a two-player zero-sum finite deterministic turn-taking game, after a suitable augmentation of the state space. The additional states formalize the alternating-turn aspect of the game, and encode the defender's foreknowledge of the attacker's choice. These additional states, however, do not require explicit representation. They do not significantly increase the amount of computation required to solve the game as addressed in section IV-B. In the remainder of this section, we will denote the attacker and defender by A and D, respectively.

In addition to the primary states in $\mathcal{Z}$ introduced in section III-B, a collection of intermediate states $\mathcal{I} = \mathcal{Z} \times \{1, \cdots, N\}$ is required, where the second component – the element of $\{1, \cdots, N\}$ – is used to signal the attacker's choice to the defender. The entire state space is thus the disjoint union $\mathcal{X} = \mathcal{Z} \sqcup \mathcal{I}$. For states in $\mathcal{Z}$, it is the attacker's turn, and his action set is $\mathcal{A}_A = \{u_1, \cdots, u_N\}$; for states in $\mathcal{I}$, it is the defender's turn, and his action set is $\mathcal{A}_D = \{r_1, \cdots, r_M\}$. Fig. 4 shows the two players taking turns. Primary states $z_{ij} \in \mathcal{Z}$ are marked by i, j. Intermediate states are shown in gray. Costs are given in section III-D.

What we seek to find now is a stationary, subgame-perfect, Nash-equilibrium pair of policies for the two players of this finite zero-sum Markov game. *Stationary* means that the (stochastic) policy is a function only of the current state and not on the time; *Nash-equilibrium* means that no player can unilaterally improve his expected reward by changing his policy; and *subgame-perfect* means that the policies remain a Nash-equilibrium when restricted to any subgame. [13]

This means two things. First, each player $P \in \{A, D\}$, must have a function $p_P : \mathcal{X} \to \mathcal{A}_k$ that, for each state $x \in \mathcal{X}$, returns the action that the agent should take in this state; and that

$$(p_A, p_D) = \arg\max_{p_A} \min_{p_D} \mathbb{E}\{C(p_A, p_D)\} \ . \tag{5}$$

In other words, we would like to find the policies that minimize the defender's worst case expected cost. By Nash's theorem for finite two-player zero-sum games, these are also the policies that minimize the attacker's worst case expected cost. This corresponds to two perfectly intelligent and conservative players, who each play the game flawlessly in order to prevent whatever outcome is worst for themselves.

D. Move Costs

As they play the game described in the previous section, the two players should attempt to make choices that are best for themselves. We define *best* by imposing a zero-sum cost structure on game dynamics. The defender seeks to minimize his cost, and the attacker seeks to maximize it.

For each $k \in \mathbb{N}$, we define the instantaneous cost

$$c_l(p_A, p_D) \triangleq J_D(j_{l-1}, i_{l-1}, j_l, i_l) - J_A(i_{l-1}, j_{l-1}, i_l, j_l) \tag{6}$$

where i_l and j_l are the values of i and j on the l-th iteration of algorithm 1 when the attacker and defender make choices according to the policies p_A and p_D respectively, and where J_D and J_A are problem-specific cost functions that describe how difficult it is for each of the players to make the corresponding moves.

Then, the total discounted cost paid by the defender is,

$$C(p_A, p_D) = \sum_{k=1}^{\infty} \gamma^k c_k(p_A, p_D) \tag{7}$$

for some $\gamma \in (0, 1) \subset \mathbb{R}$. Together with the previous section, this fully specifies the Markov game played by the attacker and defender.

IV. Model Construction and Solution

For a general hybrid adversarial problem, we construct a model based on the expected actions of the adversary and the capabilities of the defender.

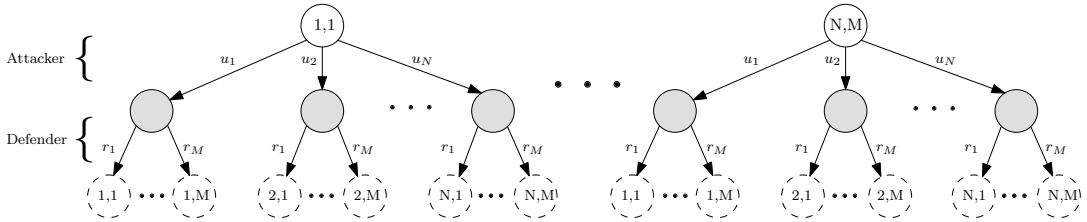


Fig. 4: Game with alternating turns

A. Constructing a Model

The attackers actions should achieve coverage over the set of possible moves. The defender's actions are chosen by considering the defender's capabilities with respect to the attacker's moves. For example, we propose that the defender's actions be explicitly optimized for each type of action by the attacker. Notice that there may be multiple defenses for a single attack. Even if one is locally suboptimal to another, it may be globally optimal in the context of an extended sequence of moves.

This can be seen as a way of making the problem feasible by subdividing it into two steps. First, we find a finite set of locally optimal responses. Second, we find the global optimum constrained by the finite set of local optima. These may be worse than the unconstrained global optimum. Response functions should be chosen in such a way that the constrained global optimum is as close to the unconstrained one as possible.

B. Solving the Markov Game

The best policy for the defender is identified by using value iteration. Any policy for the defender maps each of the intermediate states $\mathcal{I}$ to an action. The number of intermediate states is the product of the total number of primary states $\mathcal{Z}$ and the number of attacks. Value iteration is run only over primary states. We do not explicitly store values for the more numerous intermediate states. They are implicitly part of the calculation as we assume that the attacker is always choosing the worst attack for the defender. The cost of a primary state is iteratively calculated by

$$C_{i,j}^{k+1} = \min_{i'} \max_{j'} (J_D(i,j,i',j') - J_A(i,j,i') + \gamma \cdot C_{i',j'}^{k}) \quad (8)$$

$C_{i,j}^{k}$ is the cost for the primary state (i,j) at the k^{th} iteration. $J_D(i,j,i',j')$ and $J_A(i,j,i')$ are the costs of the defender and attacker, respectively, which consists of both the cost for the contact state and the cost for motions that get to the contact state. Using the computed costs for the primary states, the optimal policy for the defender over all intermediate states, i.e. over all primary states (i,j) and attacks i', is given by

$$j^*(i,j,i') = \underset{j' \in \{1,\cdots,M\}}{\arg\max} \; (J_D(i,j,i',j') - J_A(i,j,i') + C_{i',j'}) \quad (9)$$

V. ConAn: Contact Anticipation

In this section we present sword fighting as an example application of the adversarial interaction model between a human and a robot.

A. Problem Description

Given the model presented in the previous sections, the *attacker* is a human armed with a foam sword. The *defender* is a 7-degree-of-freedom robotic arm, also grasping a foam sword. The attacker's goal is to strike the defender, and the defender's goal is to repeatedly parry these attacks by deflecting them with its own sword. It should simultaneously force the attacker to make difficult moves. The following sections develop a model for this sword combat in terms using the formal machinery previously introduced.

1) Defender Model: The defender uses a kinematic motion model where motions are constrained by joint velocity and acceleration limits. In other words, the model for the defender takes the form,

$$\dot{q}(t) = v(t) \quad (10)$$

$$\dot{v}(t) = u(t) \quad (11)$$

$$\text{s.t.} \; \begin{matrix} |v^i(t)| \leq \omega_{\max}^i \\ |u^i(t)| \leq \alpha_{\max}^i \end{matrix} \quad \forall i \in \{1, \cdots, 7\} \quad (12)$$

where $x(t) \triangleq (q,v)(t) \in \mathbb{R}^7 \times \mathbb{R}^7$ is a point in local coordinates in the tangent bundle $\mathsf{T}Q$ to the mechanical configuration manifold Q; $u(t) \in \mathbb{R}^7$ is the instantaneous control input of joint torques; and $\omega_{\max}, \alpha_{\max} \in \mathbb{R}_+^7$ are joint velocity and acceleration limits, respectively. Calling the forward kinematic map for the end-effector $h : Q \to SE(3)$, one may think of $h(x)(t)$ as the output of the system at any time.

2) Attacker Model: In order to simplify the attacker motion, we assume a free-flying sword and do not model the arm of the attacker. However, we limit the motions of the attack to the surface of a sphere. There are no acceleration or velocity limits for the attacker.

We assume that there is a finite set of possible attack trajectories, which are known before the fight. At the moment the attacker starts an attack, the chosen attack is known to the defender. There exists a function that returns the configuration and velocity for any given sword point during the attack. The defender can choose the block based on this knowledge. The attacks following the current one are unknown until they are executed.

The defender should choose the best action based on the knowledge of the current attack and of the set of all possible future attacks.

B. Attacker motion

In section III-A we described the attacker motion in two stages: the actual attack and a transition motion towards the

precondition of the attack. We now describe the implementation of these two stages.

Other parts of our implementation do not rely on a specific class of attacker motions. However, in order to generate and represent the attacker motion easily, we restrict it to the surface of a fixed sphere. The attacker's sword is always located at a given distance from a fixed point and pointed away from it. This somewhat resembles a human swinging a sword.

We generated $N = 5$ attacks at even intervals. The attacks $x_1, \cdots, x_5$ stay on geodesics. This places all attack paths on circular segments. The circle is the intersection of the sphere and a plane represented by its normal vector. To identify the segment of the circle, we specify the direction of the start location and an angle giving the length of the segment. To specify the trajectory, we set a constant acceleration starting at zero velocity. Given these assumptions and an additional parameter specifying the total duration of the attack, the trajectory is completely defined.

The transition motion, which connects the state after a block to the beginning of an attack trajectory, is also restricted to the surface of the sphere. However, it is not restricted to a geodesic. To find an attacker motion on the sphere that connects the state after a block to the state at the beginning of an attack trajectory we use a cubic spline. We first connect the two states using the cubic spline in the 3-dimensional workspace. Then, we project the spline onto the surface of the sphere. As both the start and end positions and velocities are already on the sphere, we achieve a valid trajectory on the sphere connecting the two states. In our experiments, we manually selected the duration of transitions. There were no velocity or acceleration constraints.

C. Response functions

In section III-A we introduced response functions $r_1, \cdots, r_M$ for the defender. For each attack trajectory, a response function returns a point in time, which defines a state of the attacker within the attack trajectory, and a state of the defender. In the sword fighting domain the states of the attacker and defender returned by a response function must be such that the 2 swords are in contact. We define the different response functions to be the global minimum points in regard to different cost functions defined over the joint contact state of attacker and defender. This cost function only evaluates the contact state. It does not evaluate the feasibility or the cost of moving to this contact state.

$$r_j(x_i) = \arg\min_{T,y} \text{cost}_j(x_i(T)), y) \tag{13}$$

In order to find the optimal contact state, we first define a method that determines contact states in general (section V-C.1). We discuss different cost functions that evaluate contact states (section V-C.2) and a method to find the global minimum across all possibilities (section V-C.3). The search for an optimal contact state is presently too computationally expensive to be processed during online execution. However, since the set of attack trajectories is finite and known, we can calculate the values of the response function for all possible attack trajectories off-line.

Symbol	Description
T	Time, which defines the configuration of the attacker's sword.
ξ_a, φ_a	Coordinates specifying the contact point on the surface of the attacker's sword relative to the local coordinate frame of the sword. $\varphi_a = 0$ at the x-axis.
ξ_d, φ_d	Same as above on the defender's sword
α	Angle between the two swords
ϕ	Swivel angle of the robot arm

TABLE I: Parameters used to sample a contact state

1) Sampling contact states: Given the trajectory of the attacker's sword, we seek a time and arm state such that the two swords are in contact and the sign of the contact velocity is such that the swords converge. In this section we compute a state such that the contact constraints is met. We are not concerned with the quality or feasibility of the contact state given the initial sword position.

Sampling in the configuration or phase space of the defender's arm is not sufficient to solve this problem. The probability of obtaining an arm configuration that is in contact with the attacker's sword is zero. Instead, we first sample a configuration of the defender's sword in workspace such that the two swords are in contact. Second, we determine whether this sword configuration can be reached by some configuration of the robot arm. If so, we calculate the joint angles of the arm. Last, we pick joint velocities. We do not sample the joint velocities as they are not part of the optimized parameters. Instead, we directly pick the best velocities such that the two swords converge in an optimal state defined by the evaluation function.

To retrieve an arm configuration we sample its 7-dimensional parameter space. The 7 parameters are shown in Table I. T, defines the configuration of the attacker's sword. The 5 parameters ξ_a, φ_a, ξ_d, φ_d and α define the defender's sword configuration relative to the attacker's sword configuration. The last parameter ϕ represents the extra degree of freedom of the redundant arm. This parameter together with the defender's sword configuration defines the arm configuration uniquely.

Our strategy for sampling contact configurations is complete. Each contact configuration can be represented by at least one set of parameters. The contact state is calculated from the seven parameters as follows.

The transformation of the attacker's sword is a function of time $G_a(T)$. The time parameter T defines the transformation of the attacker's sword G_a. Using G_a and the five parameters ξ_a, φ_a, ξ_d, φ_d and α we can calculate the transformation of the defender's sword G_d and the transformation for the point of contact G_c:

$$G_c = G_a L_z(\xi_a) \cdot R_z(\varphi_a) \cdot L_x(r_a)$$
$$G_d = G_c \cdot L_x(r_d) \cdot R_x(\alpha) \cdot R_z(\pi - \varphi_d) \cdot L_z(-z_d).$$

R_x denotes a rotation around the x-axis and L_x a translation along the x-axis and r_a and r_d are the radii of the two swords.

We compute the end effector pose, T_e, from the pose of

the defender's sword:

$$G_e = G_d \cdot L_z(-\frac{l_d}{2}) \cdot R_y(\frac{\pi}{2}) \cdot L_z(-0.1)$$

l_d is the length of the defender's sword. Inverse kinematics (IK) is applied to determine whether an end-effector configuration can be reached and calculate the arm configuration. Due to redundant joints there may be multiple solutions. This ambiguity is resolved by a parameter ϕ, which specifies the redundant degree of freedom by the swivel angle of the arm. IK enforces joint limits but not collisions.

2) Evaluating contact states: In section V-C.1 we described how to find the space of valid blocks. In order to find locally optimal blocks, we must now evaluate the configuration and velocity of both, the attacker's and defender's swords. The criteria for optimality is defined as follows:

1) Keep the attacker at a safe distance
2) Minimize impulse on the joints
3) Maximize contact velocity, i.e. hit the opponent hard
4) Maximize angle of swords
5) Maximize distance from point of collision to end of sword

The cost function is a weighted sum of partial costs. Each of the partial costs implements one of the given goals.

$$J_D = \sum_i w_i \cdot J_D^i$$

The first evaluation function approximates the motion of the attacker before and after the contact with linear rays and then calculates the minimal distance from these rays to the defender. The partial cost is the reciprocal of the minimal distance. This quantity grows to infinity when the distance approaches zero. Such a measure is necessary because we cannot accept any block where the distance to the attacker is close to zero even if all other partial costs favor that block.

$$J_D^1 = \frac{1}{d_{min}}$$

Notice that we do not we do not check for self-collision explicitly. However, typical optimal configurations are free of self-collision due to the evaluation (1) since maintaining a safe distance to the attacker requires a stretched-out arm. Fig. 5(b) shows the optimal configuration of the arm for a particular attack without considering the distance to the attacker. The configuration is in collision. When taking the distance of the attacker into account as shown in Fig. 5(a) the block is visually more stable and the defender is not in self-collision.

Evaluation (2) is the Euclidean norm of the vector of impulses around the joints caused by the impulse applied to the defender's arm by the attacker at the point of contact. When calculating the impulse around one joint, we assume all other joints are fixed.

$$J_D^2 = ||I||_2 \quad \text{with } I_i = \xi_i \cdot ((p_c - p_i) \times ((v_a \cdot n_c) n_c))$$

where p_i is a point on and ξ_i is the direction of the i^{th} joint axis. p_c is the point of contact, n_c is the normal direction of the contact and v_a is the velocity of the attacker.

(2) favors configurations in which the contact point is close to and/or aligned with the joint axes. It also favors hitting the opponent from the side because only the part of the attacker's velocity normal to the contact is creating impulses on the joints. Fig. 5(c) shows that when not taking the impulses into account, the contact point is further away from the joints and the joints are not aligned in a way to reduce the impulses around the joint axes.

Evaluation (3) weighs configurations that have a large contact velocity between the two swords while meeting joint velocity limits.

$$J_D^3 = -v_c \quad \text{with } v_c = v_i \xi_i \cdot ((p_c - p_i) \times (-n_c)) = \frac{dp_c}{dq} v$$

$\frac{dp_c}{dq}$ is the Jacobian for the contact point p_c at the arm configuration q.

Evaluations (4) and (5) lead to blocks that are more robust and less sensitive to controller error. The most expensive configurations for (4) have swords parallel to each other and those for (5) have swords that are touching at the tip. In both cases, minor controller errors would lead to the swords not making contact at all. We use these evaluation metrics to ensure that the swords make broad area contact that would still occur even if the swords do not precisely achieve the expected states.

$$J_D^4 = (|\alpha| - \frac{\pi}{2})^2$$
$$J_D^5 = \xi_a^2 + \xi_d^2$$

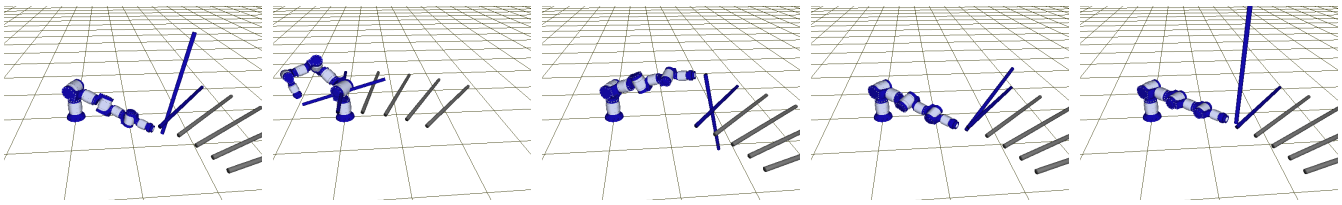
Fig. 5(d) and 5(e) shows that blocks appear more natural with metrics (4) and (5). As noted earlier, we are using different evaluation functions to retrieve different blocks to a particular attack. These different evaluation functions are structurally the same but use different weight vectors w.

In our experiments, we generated two blocks for each attack. One block was retrieved using a weight vector such that contact velocity was considered. Joint velocities were set to their positive or negative limits. The second block did not consider contact velocities ($w_3 = 0$). Joint velocities were set to zero.

3) Contact Optimization: In order to select the optimal blocks according to Section V-C.2 in the 7-dimensional parameters space, we applied the stochastic method of particle swarm optimization (PSO) [14], [15]. PSO is a population-based method for global nonlinear optimization. Its main advantages are the simplicity in implementation and performance on problems with numerous local minima. Since it is a stochastic method, it does not require the calculation of a gradient. In our experiments we ran 200 iterations on 200 particles. Optimizing for each block required approximately one minute.

D. Defender motion

Section V-C.3 calculated optimal blocks. However, it did not take into account the feasibility or the cost of achieving the blocking configuration. In section III-A we introduced a controller κ for the defender, which defines a trajectory given a start state y^0, goal state y^1 and a time T. We now



(a) baseline (b) ignoring distance (c) ignoring impulses (d) ignoring angle (e) ignoring central hit

Fig. 5: Optimal blocks are worse when leaving out one of the partial costs.

define this controller for the sword-fighting implementation. The start state corresponds to the current block and the goal state to the next block. The motion needs to meet velocity and acceleration limits for the joints and reach the new block at the given time. The time available consists of the time the attacker needs to move back to start the next attack and the point in time within the attack when the contact occurs. The contact point, in turn, depends on the next block chosen by the defender.

We neglect mass and inertia and assume that torque maps linearly to acceleration. We then search for the motion that requires the minimal amount of physical work.

Since we do not consider dynamics, we can plan a motion for each joint individually. For each joint we have start and goal angles and velocities given. We have to find a motion that connects the two states while meeting the velocity and acceleration limits. To reach this, we apply a variant of trapezoidal velocity profiles. There are three stages: during the first and third stages we accelerate or decelerate at the acceleration limit. During the second stage acceleration is zero. In standard trapezoidal profiles the start and goal velocities are zero. However, in our case both the first and second stage might be either acceleration, deceleration or mixed in the case that the velocity passes through zero. The constant velocity during the second stage might be directed in the opposite direction to the goal to achieve the goal velocity.

E. Cost function

The overall cost is the difference between the defender's and the attacker's cost. However, we do not explicitly calculate a cost for the attacker. Hence any increase in the defender's cost benefits the attacker.

The defender's cost for an action consists of the cost for the following block plus the cost for the block motion. We described different cost functions for blocks in section V-C.2. One of these assigns a cost to all blocks, even if the block was generated using a different cost function. We also ascribe a cost to each transition motion. The cost for a transition is the squared maximum velocity or infinite if the motion is not possible within the velocity and acceleration limits.

F. Results

The optimal policy calculated by the value iteration tells us the best block for each pair of previous block and current attack. This is visualized in Fig. 6. All blocks are shown as nodes. Five edges emanate from each block. Each one of the edges is directed to the optimal block for one attack. Two of the blocks, shown in grey, are never reached from any other

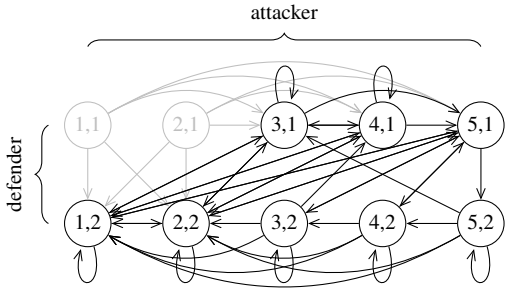


Fig. 6: Sample optimal policy computed for a given attack duration.

block, because for attacks 1 and 2 the best block is always the same independent of the previous block. For the other 3 attacks the best block depends on the previous block. Note that with the exception of the two blocks shown in grey, the graph is strongly connected. I.e. for each pair of blocks, there exists a sequence of attacks that brings us from one block to the other.

Fig. 7 shows the same set of blocks as Fig. 6. Each column shows one attack with two different defense motions. The upper row shows blocks that try to achieve a high contact velocity and the bottom row shows blocks where the arm is at rest.

There are 10 different blocks (i.e. primary states), which we can currently be in, and 5 possible attacks, which the attacker could choose next. This results in 50 intermediate states, in each of which the defender has to make a decision between two blocks. Not all of the blocks might be feasible within velocity and time constraints. In each of the 50 intermediate states the situation is one of the 5 following cases:

1) both blocks are feasible, block 1 is the better one
2) both blocks are feasible, block 2 is the better one
3) block 1 is infeasible, block 2 is feasible
4) block 2 is infeasible, block 1 is infeasible
5) both blocks are infeasible

If the attacker moves faster or the defender slower, it is more likely that one of the blocks becomes infeasible. Table II shows how often each case appears for different attacker speeds. With decreasing attack duration, increasing attack speed, more blocks become infeasible. If both blocks are infeasible for any of the five attacks, the current primary state is a death state because the attacker can choose the attack such that the defender has no options. For an attack duration of 1 s and 2 s all 10 blocks are death states, i. e. there exists an attack such that all blocks for that attack are impossible to reach from the current block. For an attack duration of

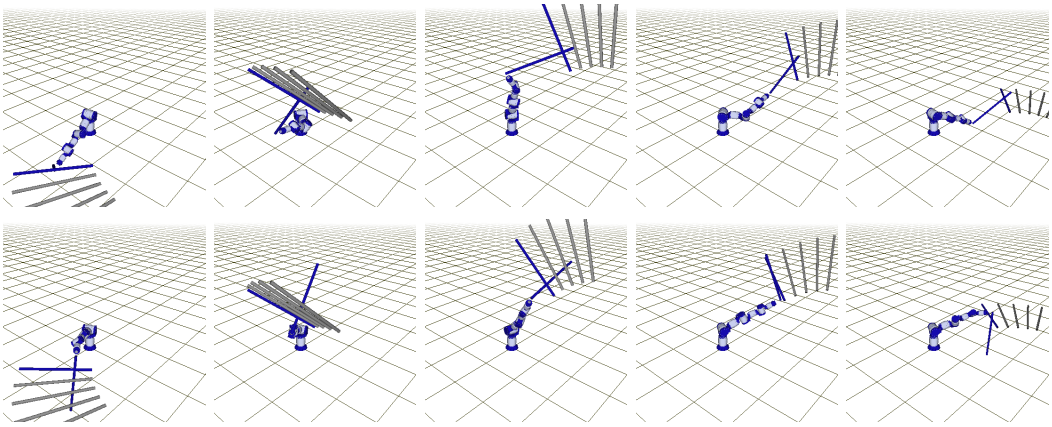


Fig. 7: Visualizations of types of blocks performed by the defender.

Attack duration	Feasible Defense: Both (Optimal Defense 1)	Both (Optimal Defense 2)	2	1	None	Death States	Time to Death
6 s	30	20	0	0	0	0	∞
5 s	37	13	0	0	0	0	∞
4 s	28	21	1	0	0	0	∞
3.2 s	1	33	8	5	3	3	0 / ∞
3 s	4	19	11	12	4	3	0 / 1
2 s	0	1	11	1	37	10	0
1 s	0	0	5	0	45	10	0

TABLE II: Defender's options for different-strength attackers

both 3 s and 3.2 s there are 3 death states. However, there is a difference: the time to death, i. e. the minimum possible number of moves the attacker needs to drive the defender from a certain state into a death state. With an attack duration of 3 s we are either already in a death state or cannot avoid transitioning to one during the next move. With an attack duration of 3.2 s the defender can always avoid death states if it does not start in one.

Fig. 6 and Table II show that both response functions are being used. It is not always the same one that becomes infeasible or is the optimal one. This result demonstrates the functionality of merging suboptimal local solutions into one global solution. A block where the defender is in rest may appear unreasonable, yet it is applied over a block with a high velocity in cases where faster velocities cannot be achieved.

The accompanying video shows the optimized combat.

VI. Conclusion

We presented a model for interaction of a robot with an adversary involving hybrid dynamics. The model is based on discretization in order to retrieve finite sets of possible actions for the attacker and defender. Motions are split into two parts, the actual attack or response and a transition motion. We validated the presented model model by applying it to simulated robot sword fighting.

Future work includes handling unknown attacks. This could be achieved by making use of a precalculated response to a similar known attack.

References

[1] R. Chipalkatty and M. Egerstedt, "Human-in-the-Loop: Terminal Constraint Receding Horizon Control," in *IEEE International Conference on Robotics and Automation*, May 2010.

[2] D. Sofge, J. G. Trafton, N. Cassimatis, D. Perzanowski, M. Bugajska, W. Adams, and A. Schultz, "Human-robot collaboration and cognition with an autonomous mobile robot," in *Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8*. IOS Press, 2004, pp. 80–87.

[3] U. C. Tools, D. Sofge, D. Perzanowski, M. Skubic, M. Bugajska, J. G. Trafton, N. Cassimatis, D. Brock, W. Adams, and A. Schultz, "Proceedings of the16th ifac symposium on automatic control in aerospace, elsevier science ltd, oxford,," in *Proceedings of IFAC Symposium on Automatic Control in Aerospace*, 2004.

[4] K. Stubbs, P. Hinds, and D. Wettergreen, "Challenges to grounding in human-robot collaboration: Errors and miscommunications in remote exploration robotics," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-06-32, July 2006.

[5] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, "Adversarial classification," in *IN KDD*. ACM Press, 2004, pp. 99–108.

[6] H. Kautz, "A circumscriptive theory of plan recognition," *Intentions in Communication*, 1990.

[7] D. E. Appelt and M. E. Pollack, "Weighted abduction for plan ascription," in *Technical Note 491, SRI International, Menlo Park*, 1992, pp. 1–25.

[8] A. G. Hofmann and B. C. Williams, "Intent recognition for human-robot interaction," in *Interaction Challenges for Intelligent Assistants*, 2007, pp. 60–61.

[9] K. A. Tahboub, "Intelligent human-machine interaction based on dynamic bayesian networks probabilistic intention recognition," *J. Intell. Robotics Syst.*, vol. 45, no. 1, pp. 31–52, 2006.

[10] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, 1994, pp. 157–163.

[11] E. A. Hansen, "Dynamic programming for partially observable stochastic games," in *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 2004, pp. 709–715.

[12] M. G. Lagoudakis and R. Parr, "Value function approximation in zero-sum markov games," in *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI 2002*. Morgan Kaufmann, 2002, pp. 283–292.

[13] G. Owen, *Game Theory*, 2nd ed. Academic Press, 1982.

[14] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*, vol. 4, 1995.

[15] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.