2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Safe and Effective Learning: a Case Study

Giorgio Metta, Lorenzo Natale, Shashank Pathak, Luca Pulina and Armando Tacchella

Abstract— **In this paper we consider the problem of ensuring that a multi-agent robot control system is both safe and effective in the presence of learning components. Safety, i.e., proving that a potentially dangerous configuration is never reached in the control system, usually competes with effectiveness, i.e., ensuring that tasks are performed at an acceptable level of quality. In particular, we focus on a robot playing the air hockey game against a human opponent, where the robot has to learn how to minimize opponent's goals (defense play). This setup is paradigmatic since the robot must see, decide and move fastly, but, at the same time, it must learn and guarantee that the control system is safe throughout the process. We attack this problem using automata-theoretic formalisms and associated verification tools, showing experimentally that our approach can yield safety without heavily compromising effectiveness.**

I. INTRODUCTION

Multi-agent robot control systems are of fundamental importance in robotics as they provide a scalable solution to support the elaboration of signals from sensors of various kinds, the execution of complex cognitive functions, and the coordination of actuators to achieve the desired goals. Indeed, most of the contemporary robotic development environments provide support for such architectures – see [1]. The intrinsic complexity of multi-agent control systems – due to the parallel execution of many interacting components – is further increased by learning agents. Pervasive in contemporary robot control systems [2], learning agents change their internal parameters over time, making it difficult, if not impossible, to anticipate all the possible behaviours of the control system as a whole. Therefore, while multiple learning agents enable robots to perform various tasks effectively, they may conceal errors which can hinder the correctness of the control system, and thus compromise the functionality of the robot.

In this paper we consider a robot control system that embeds adaptive agents. We prove their correctness against safety properties, i.e., properties requiring that a control configuration that may cause damage to the environment is never reached. Our view of safety is thus à la Formal Methods [3], wherein mathematical models of the control system and the properties that it must satisfy are used to provide formal proofs that properties hold throughout the evolution of the system. In particular, we consider an automated approach wherein the model of the system and the properties are input to a verification software which statically checks execution traces. This approach, known as Model Checking [4], has been shown to be very effective for debugging integrated circuits [5], software [6] and, more recently, also complex control systems [7]. Most of the Model Checking literature, however, deals with systems that do not include learning components. Our aim is to build on previous works in order to accommodate for (i) multi-agent control systems, (ii) learning components and (iii) a challenging setup in which they must be proved safe.

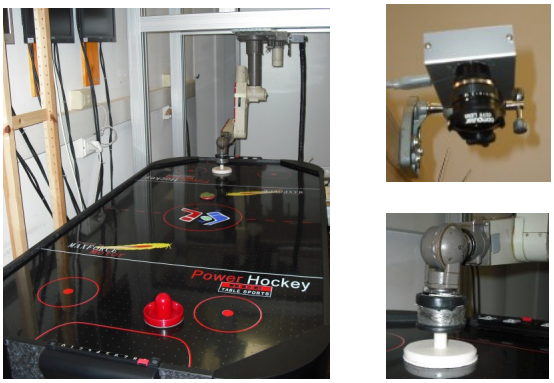

Fig. 1. The Air hockey setup (left), details of the camera (top right) and the end effector (bottom right).

Our case study involves learning to play the air hockey game with the setup depicted in Figure 1. Air hockey is a game played by two players. They use round paddles (mallets) to hit a flat round puck across a table. Air is forced up through holes in the table's surface to create a cushion of air whereon the puck slides with little friction. At each end of the table there is a goal area. The objective of the game is to hit the puck so that it goes into the opponent's goal area (offense play), and to prevent it from going into your own goal area (defense play). [1] Air hockey has already been explored as a benchmark task for humanoid robots and vision – see, e.g., [8], [9]. In the words of [8], air hockey is challenging because it is fast, demanding, and complex, once the various elements of the physical setup are taken into account. Notice that, even if our setup does not involve a humanoid robot as in [8], all the core issues are still present.

In our case study, a multi-agent control system manages the setup. A *vision* agent is devoted to visual perception;

Giorgio Metta (giorgio.metta@iit.it) and Lorenzo Natale (lorenzo.natale@iit.it) are with the Italian Institute of Technology, via Morego 30, 16163 Genova.

Shashank Pathak (shashank.pathak.iitd@gmail.com), Luca Pulina (luca.pulina@unige.it) and Armando Tacchella (armando.tacchella@unige.it) are with Università degli Studi di Genova, Dipartimento di Informatica Sistemistica e Telematica, Via Opera Pia 13, 16145 Genova.

[1] You may get a feeling of the game by watching the movie attached to this paper. The movie was obtained by collating chunks of different sessions in our experiments.

a *motion control* agent sends position commands to the manipulator; finally, a *coordination* agent converts the stimuli perceived by the vision agent into commands for motion control. The coordination agent is the only one whose internal parameters change over time for the effects of learning. In particular, the goal of learning is to fine tune coordination in order to be able to intercept the puck when it approaches the robot's goal area (defense play). We consider our control system safe if the manipulator does not reach unsafe positions, e.g., moving too close to the table's edges. Given our safety target, we model each agent as a hybrid automaton [10], a formalism allowing for mixed discrete and continuous state variables. We check execution traces for safety in a static way, by feeding the automata and the property statements to a model checker – in our case HySAT [11] – which can find bugs by exploring traces of increasing length. Because of learning, the whole system must be (re)verified eventually. We preserve safety at all times by keeping safe – and possibly ineffective – parameters of the coordination agent in place, until we have a more effective – and definitely safe – setting. While the proposed approach is expressive enough to allow for various setups to be analyzed, our experimental analysis in the air hockey setup shows that it can yield safety without heavily compromising on effectiveness. All other things being equal, the accuracy of a safety-conscious control system is very close – albeit inferior – to the one of a safety-oblivious control system.

The paper is structured as follows. In section II we describe our experimental setup, including the robot and the supporting hardware and software components. In section III we describe in detail the formal model of the robot control system, including an introduction to the formal language and the algorithms to verify safety. In section IV we comment the results of our experiments, and in section V we discuss related work.

II. SETUP

Our setup consists of a commercial air hockey table – see Figure 1 (left). The table is approximately 90cm wide and 180cm long.[2] Original features of the table and its accessories are unchanged, with the following exceptions. The red puck has been covered with a green sticker in order to make tracking easier. The mallet used by the robot is a custom-made round plastic paddle – see Figure 1 (bottom-right). The robot's goal area is covered with a metal bar that bounces back the puck so that it can be recovered without entering the robot's workspace. In the following, we describe in some detail the hardware and the software components used in our setup.

A. Hardware components

For vision, we used a CCD DragonFly2 camera with a 60Hz maximum frame rate, fitted with a fish-eye lens – see Figure 1 (top-right). The camera is mounted 145cm above the top center of the table and its mounting is not connected to the table or to the robot's scaffolding. The camera is wired to a standard PC by means of an IEEE-1394 interface. For manipulation, we used a 6 d.o.f. Unimation PUMA 260 industrial robot mounted upside down on a metallic scaffolding to match human-like arm position and movement. The robot's paddle is not rigidly fitted to its end effector, whereas a damper – see Figure 1 (bottom-right) – is used to compensate for small errors either in (i) vertical positioning, so that the robot's paddle is never pushed on the table with excessive force, or (ii) wrist orientation, so that slight changes in pitch and yaw do not alter the horizontal position of the paddle. Four standard PCs are used to provide elaboration.

B. Software agents

The whole robot control system is built on top of YARP (Yet Another Robotic Platform) [13], a middleware devoted to the integration of devices for complex robotic architectures. In our setup YARP provides seamless connection between the modules, and it enables us to distribute the components across different PCs connected to a LAN.

A vision module is dedicated to track the puck on the table and send puck positions via YARP. Our camera mounting eliminates perspective distortion – as in [9] – as well as vibrations due to robot's movements – which must be dealt with in [8] instead. However, the puck moves fast – average speeds of 800 pixels/s (about 2.5m/s) are typical – and real-time detection requires a tight compromise between speed and complexity. Our tracking algorithm of choice is based on image segmentation with background subtraction in the YUV color space. The algorithm is accurate enough for our purposes, and fast enough to acquire some (at least two) frames containing the puck inside a predefined "perception zone" of the table. As soon as the puck leaves such zone, the grabbed positions are sent to the coordination module. In our experiments, the perception zone starts at 1.1 m ($\rho\_begin\_perception$) from the robot's world origin, and ends at 0.6 m ($\rho\_end\_perception$).

A motion control module is dedicated to the low-level control of the manipulator. For the sake of speed and simplicity, control is organized around five primitives. *Home* returns the arm to its reference position. *Left* and *right*, respectively, rotate left and right the waist joint of the manipulator, so that the paddle follows a circular trajectory on the table centered in the robot's world origin. Finally, *forward* and *backward*, respectively, coordinate shoulder, elbow and the wrist bend joints of the manipulator, so that the paddle follows a linear trajectory on the table, going either outward or inward w.r.t. the robot's world origin. The above primitives are realized as follows: The angular displacement is solely controlled by the waist joint, whereas the radial displacement is achieved through constrained motion of shoulder, elbow and wrist bend, to ensure planar motion.

A coordination module is in charge of (i) reading the puck positions from vision, (ii) estimating reasonable target positions for the robot's paddle and (iii) send the esti-

[2]Our table is smaller than tables sanctioned by the United States Air hockey Association (USAA) for official tournaments, see [12], whereas the mallets and the puck meet USAA requirements.

mated position to the motion controller. In particular, (ii) is performed using a linear transformation in a system of polar coordinates centered in the robot's world origin. The estimated target position of the robot's paddle (ρ_{ee}, θ_{ee}) is the result of the following calculations:

$$\begin{array}{rcl} \rho_{ee} & = & p_1 + p_2\rho_1 + p_3\theta_1 + p_4\rho_2 + p_5\theta_2 \\ \theta_{ee} & = & p_6 + p_7\rho_1 + p_8\theta_1 + p_9\rho_2 + p_{10}\theta_2 \end{array} \quad (1)$$

where (ρ_1, θ_1) and (ρ_2, θ_2) correspond to two puck positions detected by vision, and $\mathbf{p} = \{p_1, p_2, \ldots, p_{10}\}$ are parameters learned using linear regression. Notice that coordination requires at least two points from vision in order to predict (ρ_{ee}, θ_{ee}). In the case of straight shots, two points are sufficient to compute the whole trajectory of the puck, and thus equation (1) can, in principle, compute the target position exactly. This is not the case when shots bounce multiple times, and an exact prediction would require either more parameters, or physical laws and geometrical constraints to be taken into account. However, in actual plays the puck rarely bounces more than one time on the edge, so equation (1) can still provide a good estimate of the target position, while keeping the complexity of the model low. The parameters $\mathbf{p}$ can be learned in two ways:

- **Off-line** by collecting a number of shots and related target positions, and using them as a train set to obtain the value of $\mathbf{p}$ which is unchanged thereafter.
- **On-line** by starting with some bootstrap value $\mathbf{p}_0$ and then recording new shots/target positions while playing so that new and improved values can be computed.

In our case, on-line learning amounts to adding new shots/target positions to a data set, and train linear regression from scratch. While in principle this may lead to inefficiencies and/or convergence problems, in practice the method turns out to be pretty robust and fast enough for our purposes.

III. MODELING

A. Formal preliminaries

In order to model the multi-agent control system described in the previous section we resort to the formalism of *Hybrid Automata* [10]. For our purposes, a hybrid automaton can be defined as a tuple $A = (X, V, flow, inv, init, E, jump)$ consisting of the following components. *Variables* are a finite ordered set $X = \{x_1, x_2, \ldots x_n\}$ of real-valued variables, representing the continuous component of the system's state. *Control modes* are a finite set V, representing the discrete component of the system's state. *Flow conditions* are expressed with a labeling function $flow$ that assigns a condition to each control mode $v \in V$. The flow condition $flow(v)$ is a predicate over the variables in $X \cup \dot{X}$, where $\dot{X} = \{\dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n\}$. The dotted variable $\dot{x}_i$ for $1 \leq i \leq n$ refers to the first derivative of x_i with respect to time, i.e., $\dot{x}_i = \frac{dx}{dt}$. *Invariant conditions* determine the constraints of each control mode with the labelling function inv, and *initial conditions* are denoted with the function $init$. *Control switches* are a finite multiset $E \in V \times V$. Each control switch (v, v') is a directed edge between a source mode $v \in V$ and a target mode $v' \in V$. *Jump conditions* are expressed with a labeling function $jump$ that assigns a jump condition to each control switch $e \in E$. The jump condition $jump(e)$ is a predicate over the variables in $X \cup X'$, where $X' = \{x'_1, x'_2, \ldots x'_n\}$. The unprimed symbol x_i, for $1 \leq i \leq n$, refers to the value of the variable x_i before the control switch, and the primed symbol x'_i refers to the value of x_i after the control switch. Thus, a jump condition relates the values of the variables before a control switch to the possible values after the control switch.

Intuitively, checking that a hybrid automaton is safe amounts to checking that no execution reaches an unwanted condition. In order to frame this concept precisely we define a *state* as a pair $(v, \mathbf{a})$ consisting of a control mode $v \in V$ and a vector $\mathbf{a} = (a_1, \ldots, a_n)$ that represents a value $a_i \in \mathbb{R}$ for each variable $x_i \in X$. The state $(v, \mathbf{a})$ is *admissible* if the predicate $inv(v)$ is true when each variable x_i is replaced by the value a_i. The state $(v, \mathbf{a})$ is *initial* if the predicate $init(v)$ is true when each x_i is replaced by a_i. Consider a pair of admissible states $q = (v, \mathbf{a})$ and $q' = (v', \mathbf{a}')$. The pair (q, q') is a *jump* of A if there is a control switch $e \in E$ with source mode v and target mode v' such that the predicate $jump(e)$ is true when each variable x_i is replaced by the value a_i, and each primed variable x'_i is replaced by the value a'_i. The pair (q, q') is a *flow* of A if $v = v'$ and there is a nonnegative real $\delta \in \mathbb{R}_{\geq 0}$ – the duration of the flow – and a differentiable function $\rho : [0, \delta] \rightarrow \mathbb{R}^n$ – the curve of the flow – such that (i) $\rho(0) = \mathbf{a}$ and $\rho(\delta) = \mathbf{a}'$; (ii) for all time instants $t \in (0, \delta)$ the state $(v, \rho(t))$ is admissible; and (iii) for all time instants $t \in (0, \delta)$, the predicate $flow(v)$ is true when each variable x_i is replaced by the i-th coordinate of the vector $\rho(t)$, and each $\dot{x}_i$ is replaced by the i-th coordinate of $\dot{\rho}(t)$ – where $\dot{\rho} = \frac{d\rho}{dt}$. In words, jumps define the behaviour of the automaton when switching from one control mode to another, whereas flows describe the behaviour of the automaton inside the control mode. With the concepts above, we can now define executions of the automaton as *trajectories*, i.e., finite sequences $q_0, q_1, \ldots, q_k$ of admissible states q_j such that (i) the first state q_0 of the sequence is an initial state of A, and (ii) each pair $(q_j, q_j{+}1)$ of consecutive states in the sequence is either a jump of A or a flow of A. A state of A is *reachable* if it is the last state of some trajectory. Let us assume that a safety requirement can be specified by defining the "unsafe" values and value combinations of the system variables. A state is thus safe if it does not entail such values, and the whole system is safe exactly when all reachable states are safe. Safety verification, therefore, amounts to computing the set of reachable states.

Given the expressiveness of the above formalism, it is no surprise that checking safety is, in its most general form, an undecidable problem in hybrid automata [10]. Even if the air hockey control system can be modeled in terms of a *linear hybrid automaton*, i.e., a hybrid automaton where the dynamics of the continuous variables are defined by linear differential inequalities, there is still no guarantee that the exploration of the set of reachable states terminates. The method is still of practical interest, however, because

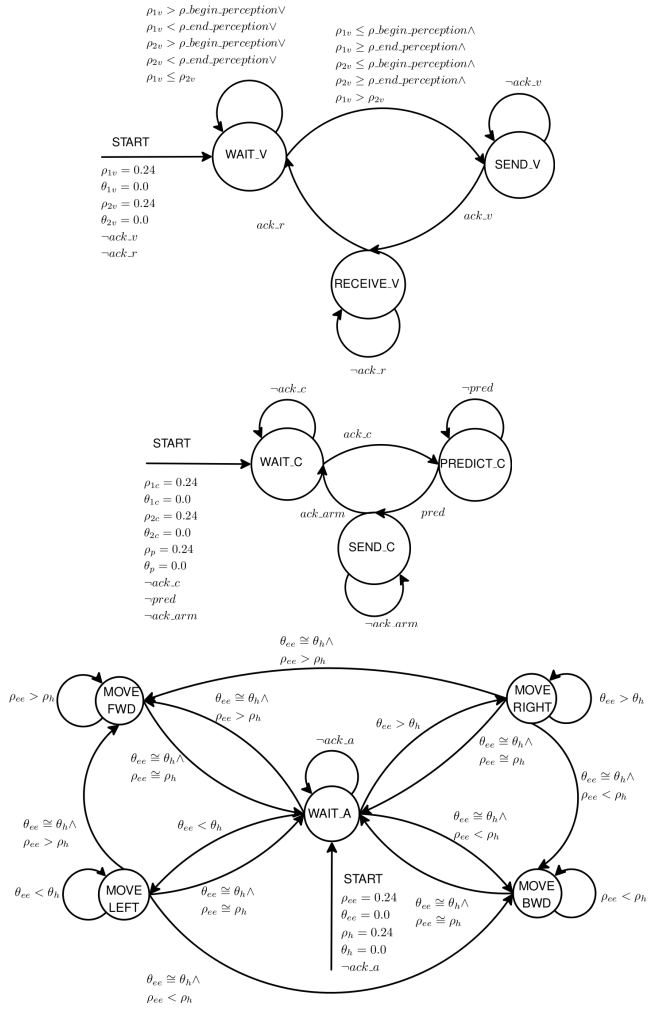

Fig. 2. Vision (top), coordination (middle), and motion (bottom) automata. Control modes are vertices, and directed edges are control switches whose labels represent jump conditions. Looping edges are labeled with invariant conditions. Initial conditions and initial values of the variables are shown as labels of incoming arrows (START). Symbols have the usual meaning, with the exception of "$=$" which indicates assignment, and "$\cong$" which indicates approximate equality testing: $x \cong y$ iff $x = y \pm \delta$, where δ is a small positive constant.

terminations can be enforced by considering the behavior of the system over a bounded interval of time. This technique, known as Bounded Model Checking – see, e.g., [11] – involves the exploration of increasingly long trajectories until either an unsafe state is reached, or resources (CPU time, memory) are exhausted. Technically, we no longer speak of verification, which is untenable in an infinite state space, but of falsification. Whenever an unsafe state is found, then we have a trajectory witnessing the bug. On the other hand, the unfruitful exploration of increasingly long trajectories is considered an empirical guarantee of safety.

B. Control agents as hybrid automata

In order to check the safety of the control system described in Section II, we model each agent as a hybrid automaton, and then we check the composition of the agent's automata. All the automata that we describe in the following are depicted in Figure 2 (see the legend for graphical conventions).

The vision automaton – Figure 2 (top) – has three control modes: WAIT_V, SEND_V, and RECEIVE_V. The real variables are (ρ_{1v}, θ_{1v}) and (ρ_{2v}, θ_{2v}) denoting the two positions perceived along the puck trajectory by the camera. The Boolean flags ack_v and ack_r are used for communications with the other agents. Initially, the automaton is in the control mode WAIT_V until at least two points are perceived along the trajectory of a shot directed towards the robot's goal area. When the two points are collected, the control mode SEND_V is reached, and the automaton loops there while the ack_v signal is false. When the coordination automaton acknowledges receipt of the coordinates, ack_v becomes true, and the mode RECEIVE_V is entered. The automaton resets to WAIT_V as soon as the motion controller also receives its target position and ack_r becomes true.

The coordination automaton – Figure 2 (middle) – has three control modes: WAIT_C, PREDICT_C, and SEND_C. The real variables are (ρ_{1c}, θ_{1c}) and (ρ_{2c}, θ_{2c}), corresponding to the puck coordinates received by the vision automaton; the variables (ρ_p, θ_p) represent the result of applying equation (1) where $(\rho_1, \theta_1) \equiv (\rho_{1c}, \theta_{1c})$ and $(\rho_2, \theta_2) \equiv (\rho_{2c}, \theta_{2c})$. The Boolean flag $pred$ denotes the end of the computation, and the Boolean flags ack_c, and ack_arm are used for communication with other agents. Initially, the automaton is in the control mode WAIT_C until the puck coordinates are received from the vision automaton. In this case, the control mode PREDICT_C is reached, and the variables ρ_p, θ_p are computed according to equation (1). Once this operation is complete, there is a jump to the control mode SEND_C, in which the coordinates (ρ_p, θ_p) are sent to the motion controller automaton. The automaton loops in SEND_C until motion control acknowledges, and then it resets to WAIT_C. Notice that we do not model the learning component of coordination as an automaton, and we consider the parameters $\mathbf{p}$ as constants. However, we can keep into account changes in $\mathbf{p}$ by scheduling (re)verification of the control system automaton each time $\mathbf{p}$ changes because of learning. In this way, the internals of the learning algorithm need not to be modeled, but it is still possible to ensure safety of the control system, as long as only safe choices of the parameters $\mathbf{p}$ are used in the coordination module.

The motion controller automaton – (Figure 2, bottom) – has five control modes: WAIT_A, MOVE BWD, MOVE FWD, MOVE LEFT, and MOVE RIGHT, corresponding to the control primitives. The real variables (ρ_{ee}, θ_{ee}) are the target coordinates of the end effector; (ρ_h, θ_h) denote the current position of the end effector. The Boolean flag ack_a is devoted to communications with other agents. Initially, the automaton is in control mode WAIT_A until the coordinates (ρ_{ee}, θ_{ee}) are received from the coordination automaton. If ρ_h or θ_h are different from ρ_{ee} and θ_{ee}, respectively, a jump condition for the control modes representing the motion primitives is satisfied, and the automaton evolves until the target position is reached. Inside the primitives, the evolution of (ρ_h, θ_h) is modeled using linear differential equations expressing constant radial and angular velocities control,

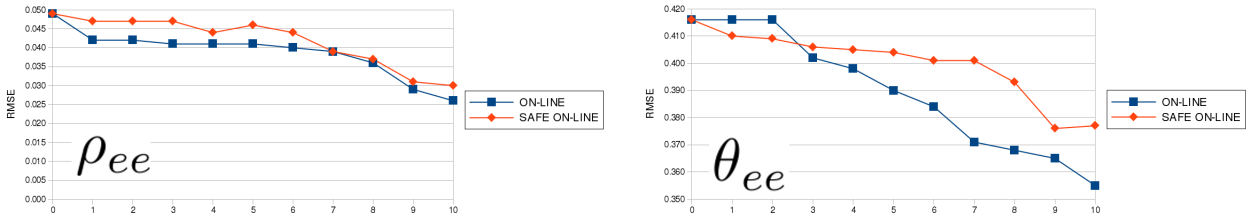

Fig. 3. RMSE on the prediction of ρ_{ee} and θ_{ee}. On the x axis the sequence of subjects, in the same order in which they played the games.

respectively.

IV. EXPERIMENTAL RESULTS

In this section we show empirical results supporting the claim that our multi agent control system can learn from experience while maintaining safety. Our safety target requires that the angle θ_{ee} is in the range $[-50, +50]$ (degrees) and ρ_{ee} is the range $[0.10, 0.27]$ (meters). These boundaries ensure that the paddle will never hit the borders of the table. Considering that the robot moves, e.g., left and right, at an average speed of 5.2 rad/s (298 degrees/s) hitting the borders would surely damage the table or the paddle. In our experiments this is prevented by a low-level emergency protection that avoids dangerous commands to be executed by the robot. In the following, when we say that the control system reached an unsafe state during experimental plays, it means that the low-level protection intervened to avoid damage. In order to check for safety we used the tool HYSAT [11], a satisfiability checker for Boolean combinations of arithmetic constraints over real- and integer-valued variables which can also be used as a bounded model checker for hybrid systems. HYSAT takes as input a textual description of the automata in Figure 2 , and it tries to find a violation of the stated safety properties by exploring trajectories of increasing bounded length k.

Data was collected by having the robot play a series of games against ten different human players[3] and using three different settings of the coordination module. In the first setting (off-line), the parameters $\mathbf{p}$ are learned off-line using a controlled training set of 150 shots performed by one of us, of which 50 are straight shots, and 100 are single-bounce shots; this setting is not checked for safety. In the second setting (on-line), the parameters $\mathbf{p}$ are learned on-line; the bootstrap parameters $\mathbf{p}_0$ correspond to a hand-made setting which is checked off-line for safety. In the third setting (safe on-line), each time a new set of parameters is learned, it is plugged into the model which is then checked for safety; the new parameters are used thereafter only if HYSAT could not find a safety violation within 30 CPU seconds. Notice that in the two on-line settings we keep learning across different players, so the more games are played, the more effective the robot becomes. Because of this, the training set accumulates the shots/target positions recorded during all the playing history. The choice of 30 CPU seconds as a resource limit for HYSAT reflects a tradeoff between the accuracy of the safety check, and the performances of safe learning. In our application, 30 CPU seconds are sufficient to perform a number of iterations k in the order of hundreds. Considering that most bugs can be found withing a limited number of iterations – k in the order of tens, running time of a few seconds – this provides a sufficient empirical guarantee of safety.

[3]At `http://www.liralab.it/airhockey` we made available the videotapes of all the experimental sessions.

TABLE I
UNSAFE PREDICTIONS DURING THE EXPERIMENTS.

PLAYER	OFF-LINE		ON-LINE	
	SHOTS	UNSAFE	SHOTS	UNSAFE
# 1	59	–	55	1
# 2	56	2	72	3
# 3	46	1	39	–
# 4	61	–	46	–
# 5	58	–	80	–
# 6	48	–	69	–
# 7	84	6	76	1
# 8	44	2	84	–
# 9	103	–	112	–
# 10	99	8	86	–

The conclusions of our experiments are that

- the off-line setting is unsafe;
- on-line learning, even starting from a safe choice of parameters, improves performances over time, but it hinders safety;
- safe on-line learning can be almost as effective as its counterpart, and it guarantees safety.

In more detail, considering the parameters $\mathbf{p}$ used in the off-line setting, HYSAT is able to detect an unsafe state within 5 CPU seconds and $k = 10$. On the other hand, the set of boostrap parameters $\mathbf{p}_0$ used in the on-line settings could not be found unsafe after 10.42 CPU hours and $k = 9480$.[4] One may ask how significant it is in practice that the off-line setting is unsafe. In Table I we show, for each player, the number of shots detected (SHOTS) and the number of shots that triggered the low level protection (UNSAFE). As we can see, 50% of the players performed shots such that the control system proved to be unsafe in the off-line setting.

A similar problem occurs in the on-line setting. Looking at Table I, we can see that coordination emits unsafe target positions in five cases. Interestingly, four cases occur with

[4]Experiments with HYSAT are performed on a family of identical Linux workstations comprised of 10 Intel Core 2 Duo 2.13 GHz PCs with 4GB of RAM.

the first two players, meaning that the very first attempts to update the initial safe model invariably result in unsafe ones. One case occurs also for player #7, indicating that, no matter how effective the model becomes, still there is chance to emit unsafe predictions. In the on-line safe setting, 879 shots were detected, and none of them triggered an unsafe target prediction. In order to evaluate effectiveness, for each player we consider the parameters $\mathbf{p}$ obtained at the end of the corresponding session, so that we can compute (ρ_{ee}, θ_{ee}) with equation (1). To measure the effectiveness of the parameters $\mathbf{p}$, we extract input coordinates and reference target positions from the off-line training set, and we compute the root mean square error (RMSE) between the reference target positions and the results of equation (1). This gives us an idea about how much the control system is effective and how much it improves across the sequence of plays. In Figure 3 we show the RMSE values in for ρ_{ee} (left) and θ_{ee} (right). As we can see, both on-line settings are able to improve their effectiveness while playing. The basic on-line setting can actually outperform the safe on-line setting. However, we know it is unsafe. On the other hand, in the safe on-line setting, adaptation is slower, and the final target RMSE is higher. However, the differences between the final RMSE values is small – 13% of the on-line RMSE for ρ_{ee} and 6% of the on-line RMSE for θ_{ee} – , whereas the resulting control system is safe.

V. DISCUSSION AND RELATED WORKS

Our results show that verifying the safety of a multi agent control system with learning components is doable, even under the tight constraints imposed by real world applications. While ensuring formal safety of the control system is just one of the tasks involved in robot's functional safety – see, e.g. [14] – if a robot is to be deployed in a context where it can harm humans, it is expected that its architecture and implementation reach the highest safety integrity levels (SILs). In most cases, achieving such levels entails the use of structured design methods – such as hybrid automata – and the provision of formal guarantees of safety for the implementations – such as Model Checking or other verification methods. Therefore, we see our approach as one of the key elements towards reaching industry-standard safety for robots operating outside structured environments.

The issue of coping with formal safety in robot control systems has been posed already. The ICRA workshop "Formal Methods in Robotics and Automation" – see, e.g., [3] for the latest event – focuses precisely on this theme. However, to the best of our knowledge, this is the first time in which the issue is investigated in the context of a fast-paced task like the air hockey game, without using simulation results only, and including adaptive components. For other works in robotics dealing with the issue of safety, but not with the problem of formal verification, see, e.g., [15].

Some work has been done in the AI community towards providing a formal framework for checking the safety of adaptive agents. Relevant contributions include a series of paper by Gordon, see e.g. [16]. In these contributions, adaptation is explored as a challenge for safety, but in all of them the formal model of the system does not include continuous state variables, which makes them unsuitable for the kind of modeling required by physical robots. A rather different approach in exploring the same challenge is taken in [17] where the problem of learning how to combine different control modes in a safe way is studied. While this approach is suitable for physical robots, it requires knowledge of the systems' models to be applied. In our case, it is exactly the systems' model which is sought by learning, which makes the approach in [17] inadequate.

ACKNOWLEDGMENTS

This research has received funding from the European Community's Information and Communication Technologies Seventh Framework Program [FP7/2007-2013] under grant agreement N. 215805, the CHRIS project.

REFERENCES

[1] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007.

[2] J. Bagnell and S. Schaal, "Special issue on Machine Learning in Robotics (Editorial)," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 155–156, 2008.

[3] G. Pappas and H. Kress-Gazit, Eds., *ICRA Workshop on Formal Methods in Robotics and Automation*, 2009.

[4] E. Clarke, O. Grumberg, and D. Peled, *Model checking*. Springer, 1999.

[5] C. Kern and M. Greenstreet, "Formal verification in hardware design: a survey," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 4, no. 2, pp. 123–193, 1999.

[6] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, "Model checking programs," *Automated Software Engineering*, vol. 10, no. 2, pp. 203–232, 2003.

[7] E. Plaku, L. Kavraki, and M. Vardi, "Hybrid systems: From verification to falsification," *Lecture Notes in Computer Science*, vol. 4590, p. 463, 2007.

[8] D. Bentivegna, C. Atkeson, and G. Cheng, "Learning tasks from observation and practice," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 163–169, 2004.

[9] B. Bishop and M. Spong, "Vision-based control of an air hockey playing robot," *IEEE Control Systems Magazine*, vol. 19, no. 3, pp. 23–32, 1999.

[10] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," *Lecture notes in computer science*, pp. 209–229, 1993.

[11] M. Franzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, "Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 1, pp. 209–236, 2007.

[12] "United States Air hockey Association web site," Last visited [9-2009].

[13] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: yet another robot platform," *International Journal on Advanced Robotics Systems*, vol. 3, no. 1, pp. 43–48, 2006.

[14] D. Smith and K. Simpson, *Functional safety: a straightforward guide to applying IEC 61508 and related standards*. Butterworth-Heinemann, 2004.

[15] E. Cervera, N. Garcia-Aracil, E. Martinez, L. Nomdedeu, and A. del Pobil, "Safety for a robot arm moving amidst humans by using panoramic vision," in *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008*, 2008, pp. 2183–2188.

[16] D. Gordon, "Asimovian adaptive agents," *Journal of Artificial Intelligence Research*, vol. 13, pp. 95–153, 2000.

[17] T. Perkins and A. Barto, "Lyapunov design for safe reinforcement learning," *The Journal of Machine Learning Research*, vol. 3, pp. 803–832, 2003.