Noname manuscript No.
(will be inserted by the editor)

Event-Based Automated Refereeing for Robot Soccer

Danny Zhu · Manuela Veloso

the date of receipt and acceptance should be inserted later

Abstract The RoboCup Small Size League (SSL) is a robot soccer game with robots that play on a customized field with overhead cameras. The majority of the research effort to date has been on the performance of the autonomous teams in aspects of motion planning and team strategy. However, another critical component of a robot game is the referee. In current SSL games, refereeing is done by humans, who use a "referee box" that passes their calls to the robots. In this work, we contribute an automated referee (autoref) for SSL games, towards enabling games to proceed with little or no human supervision. The goal is to move closer to the eventual full automation of complete games with real robots. The technical challenges include the clear definition of the rules of the game in terms of features to be extracted from the visual perception, temporal sequencing, and corresponding calls and game management. We provide a description of a game of SSL as it is relevant to an autoref, by categorizing the rules of the game and presenting the structure of a game as a hybrid automaton. We then describe the complete autoref using a modular event-based architecture, following up on the automaton as a guideline, to keep track of the state of a game and issue referee commands accordingly. We present the results of using our autoref to referee games on real robots, as well as a comparison of the events detected by the autoref to the calls made by a human referee during the real SSL games at RoboCup 2014.

Keywords RoboCup SSL · hybrid automata · event detection · game formalization

D. Zhu · M. Veloso
Computer Science Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15232, USA
E-mail: dannyz@cs.cmu.edu

1 Introduction

In the most general form of our research, we address the challenge of creating an intelligent agent to observe data produced by a dynamic continuous process in order to find when the data matches some specific predefined patterns. In addition, the agent needs to make decisions about how to proceed based on the data patterns detected.

Often, the decisions that need to be made and the patterns that need to be matched at any given point in time fall into discrete categories; meanwhile, the process contains components that evolve continuously over time. For instance, an agent tasked with monitoring traffic at an intersection might be concerned with the state of the traffic light and number of cars in view (discrete), as well as the current time and the velocities of the cars (continuous). This combination of discrete and continuous elements defines a hybrid system. Accordingly, such agents can often be modeled using some representation designed for hybrid systems; one common choice is a hybrid automaton, which we explored in this work.

We studied this general research problem in the context of automated refereeing for the RoboCup Small Size League (SSL), which is described in detail in Section 3; in brief, it is intended to be analogous to a scaled-down version of FIFA soccer that uses robots as the players. A challenging example of the general problem is the creation of an automated referee (autoref) for a game of robot soccer. Autorefs have long been in use for robot soccer games in simulation, but no such referees exist yet for real robots. The soccer games with real physical robots are refereed by humans.

Our motivation for developing an autoref specifically is twofold. Practically speaking, the software behind our competitive team has grown quite complex over the

years, and it is often unclear what effect new changes will have on our overall performance. We have historically done very little quantitative testing of our code as a whole, since the need for refereeing means that it takes a great deal of time and attention to evaluate its performance in depth. Having an autoref means that it is possible to run large amounts of game time with relatively little human intervention — and none at all in simulation. Apart from that benefit, we also simply saw this as an important goal for RoboCup. Almost all effort to date has gone into creating the competitors in the various leagues, but automated refereeing is a closely related and also highly desirable goal.

Initially, all refereeing of the real robots was done manually by humans, including the positioning of the robots to restart the game after a foul or goal. But we soon introduced the "referee box" to the RoboCup SSL: a computer program that connects the human refereeing to the robots, by having a human select any of all the possible conditions that can occur in a game, and then communicating that condition to the physical robots. The robots then act accordingly and autonomously position themselves in response to the command received. Adding an autoref takes the next step by making the selection itself autonomous, fully closing the loop of the robotic behaviors.

While we ground our work on the concrete RoboCup SSL game, we contribute a general formalism and algorithms to detect events from a world model provided by perception, which could be achieved for different games. Our autoref is suitable for the refereeing of a robot game, with a set of rules dependent on perceptual features, and corresponding calls.

The structure of this paper is as follows: In Section 2, we describe existing work related to the different aspects of our refreeing formalism and system. In Section 3, we describe the SSL and define the problem of creating an autoref for it. In Section 4, we provide a more detailed breakdown of some of the rules of the SSL as they relate to an autoref. In Section 5, we discuss the mechanisms by which input data are processed to provide a coherent view of the world to the software. In Section 6, we describe the autoref itself, in terms of a formalization of a game of SSL as a hybrid automaton and a modular architecture that effectively implements it. Finally, in Section 7, we discuss the results of some quantitative experiments, enabled by the autoref, which compare two versions of our team against each other. Readers who are familiar with the SSL may wish to proceed to Section 3.6, our definition of the interface to a referee, and then to Section 6.

2 Related work

Vail et al (2007) investigated using conditional random fields for activity recognition of different robots based on a stream of their positions over time, much like the data available in the SSL. For automatic refereeing, it is also necessary to perform recognition of a sort: a referee's fundamental task is to interpret the situations described in the rules and determine when they are happening. However, most of the relevant situations for refereeing are amenable to simple handmade detectors. More advanced techniques for classification may be useful for more difficult situations, such as those that are based on attributing intent or negligence to the teams.

Many other authors have addressed the problem of online processing of RoboCup games, in pursuit of various aims. A simple example of such an aim is the automated cameraman for the RoboCup Middle Size League (MSL) described by RFC Stuttgart (Käppeler et al 2010). The cameraman shares information with a team to keep a camera pointed toward an object of interest at all times during the game. The object of interest is usually the ball, but may instead be a goal or goalie when a robot makes a shot on goal.

A more involved task is producing engaging, human-like commentary of games in real time. Rocco (Voelz et al 1999) and Mike (Tanaka et al 1998), both commentators for simulation games, use hierarchical declarative event systems to produce discrete events which may be the topic of utterances. CMCast (Veloso et al 2008) is a commentary system for soccer games played by Sony AIBO robots; it is embodied in two humanoid robots which can autonomously move around the field, localize themselves, track the ball, detect game events, and give commentary appropriately. All of these commentators revolve around a concept of discrete "events" derived from the continuous state of the game, similar to the event detectors described later in this paper.

Commentating requires more understanding of the specific game being observed than does the cameraman, but less than full refereeing; it shares the problems of event detection with refereeing, but the focus of such systems is typically more on utterance selection and audience interaction.

There have also been other efforts to provide automated refereeing for RoboCup games. For many years, the simulation leagues have had official autorefs integrated with their simulation servers, which have grown to be encompass many rules at this point, although games still require human referees for the more subjec-

tive events. In both the 2D (Chen et al 2003) and 3D[1] simulation leagues, the autorefs can handle

- managing the length of game halves,
- teleporting robots to legal positions when the game is stopped,
- detecting when a free kick is taken,
- awarding the proper kicks when the ball leaves the field or enters a goal, and
- detecting offsides situations.

Additionally, the 2D referee can detect backpasses, but relies on a human referee to detect players surrounding the ball or a team blocking its own goal with too many players. The 3D referee can detect when the players have formed a cluster around the ball that should be broken up and when a team's goal is being blocked by too many of that team's robots; in either case, it teleports the offending robots outside the field.

The autoref we describe in this paper, when running with a simulation of real robots (Section 3.3), is similar to these referees in scope. However, referees that are designed purely for simulation and execute within the simulator itself have the advantage of being able to rely upon having full, direct access to the state of the world, in terms of both reading and writing. As a result, detection of many events is much less error-prone for them.

For real robots, Arenas et al (2009) developed a refereeing robot that watched games of the RoboCup 2-Legged Standard Platform League and the Humanoid League. By necessity, that work mainly focused on the lower-level aspects of event detection from vision, since those leagues do not have the standard centralized vision system of the SSL, but the ultimate aim is equivalent to ours for those leagues. For the MSL, Tech United Eindhoven (Schoenmakers et al 2013) briefly described an autoref that can detect when the ball leaves the field, automatically signal the robots to stop play, and wait for a human to place the ball in the correct position. Their algorithm is like a simplified version of the one described here. The RoboCup Logistics League (RCLL) has also recently gained an autoref (Niemüller et al 2013), which was initially developed at the same time as the work described here. The kinds of events that need to be detected for that league are rather different, but the ultimate goals described there are much the same: taking a difficult job off human referees, ensuring objectivity in judging, and benchmarking the systems used in the game. The RCLL autoref communicates with team robots and neutral machines over the network, awarding points when robots complete particular tasks using the game pieces, which abstractly represent materials in a factory. The communication the autoref receives consists of discrete messages, primarily about completed tasks, so that it does not need to perform its own processing of real-world data to extract relevant events, as ours does. However, the rule-based system at the core of the RCLL autoref achieves a similar purpose to the structures described here, and could be used as the basis of an alternate implementation.

Finally, at least two existing projects have worked toward automatic refereeing for the SSL specifically. At RoboCup 2014, the SSL released the technical challenge of detecting certain relevant events from the game, such as robots colliding with excessive force or kicking the ball too fast[2]. ER-Force, the SSL team of University of Erlangen-Nuremberg, has published the code it developed for the challenge[3]. Also, the open-source project `ssl-autonomous-refbox`[4] uses Prolog to declaratively define the rules as predicates over game states. The ER-Force code can only handle detecting those isolated events, while `ssl-autonomous-refbox` is limited to observing a game and lacks the ability to handle the transitions related to restarting the game after a stop; this work provides a more complete solution than either of them by being able to handle a full game from start to stop, transmitting the appropriate commands to teams.

Hybrid systems are widely found in robotics applications, due to the interaction between discrete computers and the continuous physical world. Hybrid automata, a standard representation of hybrid systems, originated in the field of formal verification (Henzinger 2000; Alur et al 1993), and are typically used in that capacity. They also find use as models for robotic systems more generally (Egerstedt 2000; Srinivasa et al 2012), including concurrent multi-robot systems (Fierro et al 2001; Klavins and Koditschek 2000), but they are generally used to model either the internal components of either one monolithic agent or the behaviors of multiple cooperating agents. The model described here incorporates the observation of robotic entities that are outside the direct control of the agent itself in order to model an abstract system that arises from all of the robots together.

3 The RoboCup SSL

In this section, we provide some background information about RoboCup in general, and the RoboCup SSL in particular, including the physical design of robots, the

[1] http://simspark.sourceforge.net/wiki/index.php?title=Soccer_Simulation&oldid=3043

[2] http://robocupssl.cpe.ku.ac.th/robocup2014:technical_challenges

[3] https://github.com/robotics-erlangen/autoref

[4] https://code.google.com/p/ssl-autonomous-refbox

interaction between the various hardware and software components involved, and the general flow of a game over time. The rules of the game are described in detail in Section 4.

3.1 RoboCup and the SSL

The RoboCup SSL promotes research in multi-agent coordination in real-world adversarial domains. In this league, teams of six robots play a scaled-down version of FIFA soccer with a golf ball in a field of size 9 m × 6 m. Each robot is controlled via radio commands sent by its team's offboard computer. Four overhead cameras observe the field and detect the positions of the robots and ball with high frequency and fidelity; as compared to other RoboCup leagues, the global vision and centralized planning mean that teams focus on hardware development, coordination behaviors, and high-level teamwork strategy, as opposed to perception, locomotion, or methods of distributed planning. Games in the SSL are fast-paced, with the robots traveling up to 3 m/s and kicking the ball at speeds of up to 8 m/s.

The overarching target of the RoboCup Federation, which includes several other soccer leagues, based on widely varying robot types, as well as some non-soccer competitions, is[5]

> By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup.

The various leagues are intended to collectively spur development in the different areas of development that will ultimately be necessary to produce such a team. The humanoid soccer leagues, for example, focus on bipedal walking and ball manipulation; the SSL focuses on centralized planning and fast reactions in a dynamic environment; the MSL, which uses roughly human-sized wheeled robots and a normal soccer ball, focuses on decentralized planning and sensing without dealing with the difficulties of legged robots; finally, the simulation soccer leagues (2D and 3D) use simplified models of the world to allow teams to focus more on strategy and artificial intelligence without having to maintain hardware.

3.2 Physical components

Although there are few restrictions on the physical design of SSL player robots, except that each one must fit within a cylinder of diameter 180 mm and height 150 mm, most teams have converged toward very similar designs; Figure 1 demonstrates a typical design[6], and Figure 2 shows a typical SSL match at the RoboCup tournament. The robot has an omnidirectional drive system based on four omniwheels. For ball manipulation, it has a dribbler, a horizontal rubber-coated bar which rotates to put backspin on the ball, allowing the robot to hold the ball for some time while driving around. Finally, the robot has both a flat kicker, which kicks the ball straight forward at high speed, and a chip kicker, which kicks the ball at an angle of about 45° above horizontal. Use of the chip kicker enables robots to pass or clear the ball even when there are opponents in front of them.

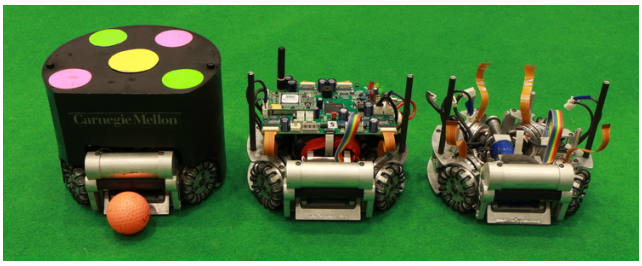


Fig. 1 The robots of CMDragons, showing (left) a full robot with a ball on its dribbler, (middle) a robot without its top cover, and (right) without its main electronics board

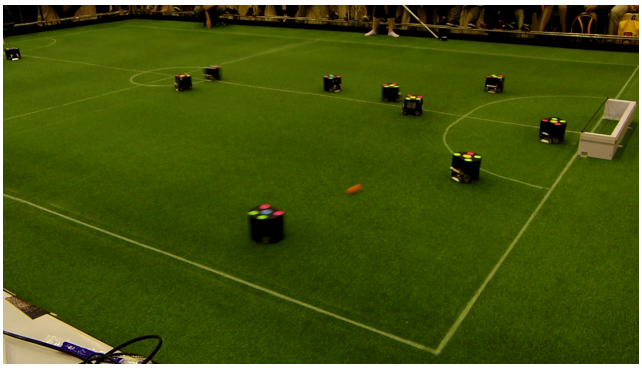


Fig. 2 A representative moment from the RoboCup 2015 tournament, showing an indirect free kick taken by CMDragons

The ball is a typical golf ball, except that it is colored bright orange for ease of detection by the vision system. According to the rules, it should be approximately 43 mm in diameter and have a mass of approximately 46 g.

The overhead cameras are mounted on scaffolding constructed around and above the field. The league has standardized on FireWire 800 cameras. All of the cam-

[5] http://www.robocup.org/about-robocup/objective/

[6] We thank Mike Licitra for the mechanical design of the robots and for the design and fabrication of the robot electronics.

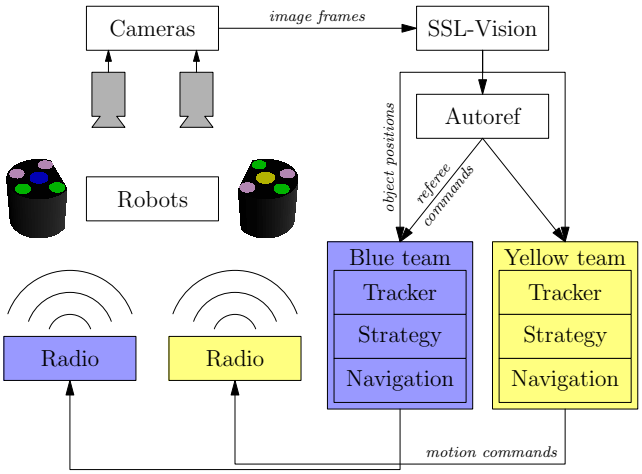


Fig. 3 A schematic diagram of the components of an SSL game. The two teams, their robots, and the neutral cameras together make up one largely autonomous system

eras are connected to a neutral computer running the SSL Vision System (SSL-Vision) (Zickler et al 2009), which processes the images from the cameras to extract the positions of the robots and ball on the field. Each camera transmits sixty frames per second to SSL-Vision; each time a frame is received from any camera, SSL-Vision broadcasts the detected positions from that camera to the team computers.

The referee's decisions are transmitted to the teams via another computer running a specialized program, which is further described below.

To ensure low latency and avoid bandwidth constraints between the team computers, referee computer, and vision computer, all of them are connected via a wired Ethernet network specific to the playing field.

Figure 3 shows a simplified overview of the overall connections between the components.

3.3 Simulation

The separation enforced by the network means that individual components can be switched out without disrupting the rest. Of course, this ability is used in playing different pairs of teams against one another; it doesn't matter to one team if the other team is replaced by a different one. But SSL-Vision itself can also be replaced by something else. CMDragons, and many other teams, often use a soccer simulator to fill in that place. Our simulator uses Nvidia's PhysX engine[7] to realistically simulate physical robots and a ball reacting to soccer commands (Zickler 2010).

[7] https://developer.nvidia.com/physx

3.4 Human refereeing

Currently, the rules of the game are enforced much as they are in FIFA soccer: a human referee and assistant referee watch the game, constantly checking whether any rule has come into effect. The referee signals for a stoppage by blowing a whistle, and verbally calls out the subsequent state of the game (e.g. "indirect free kick for yellow"); a human at a referee computer near the field then enters the commands into a specialized referee box program (refbox), which transmits them to both teams. Figure 4 depicts the refbox interface. The commands and their meanings are defined by the league's technical committee[8]; some of the most commonly used ones are described in Table 1. If `TEAM` appears in the name of an entry there, it actually corresponds to two commands, one for the blue team and one for the yellow team. The refbox and the predefined commands make up the primary interface between humans and robots in the game, and provide a powerful and efficient way to transfer the necessary judgments to the teams.

Apart from announcing the calls, the most important job of the human referees is to position the ball for each kick. The rules specify a point from which each free kick must be taken; the referees use sticks to position the ball appropriately while the game is stopped.

3.5 Game structure

Like in human soccer, the period of normal gameplay consists of two halves; in the SSL, each half consists of ten minutes of play. Game time is broken up into episodes of active gameplay punctuated by times when the game is temporarily stopped.

During an episode, the players may interact with the ball, more or less at will, and they attempt to do so in order to get it into the opponent team's goal. An episode ends, and play stops, when the ball exits the field or a player commits a foul.

When play stops, the rules specify the type of kick with which to start the next episode and a point from which the kick should be taken. While play is stopped, the robots must move at less than 1.5 m/s and remain at least 500 mm away from the ball, so that the referee may position the ball. Once the ball is in place, the referee will typically wait up to a few seconds for the robots to reach a steady state, and then call out the appropriate signal to the human at the refbox. (Waiting for the robots in this way is not specified in the rules, but is, in our experience, done to some degree by every human referee.)

[8] http://robocupssl.cpe.ku.ac.th/referee:protocol

Fig. 4 The interface displayed by the refbox when it starts up. Most of it is taken up by buttons corresponding to all the possible commands, which are automatically activated when it might be appropriate to transmit them; the refbox also keeps track of timing and game state information to act as an aid to the human user

Most kinds of kicks are to be taken by a specific team; until a member of that team takes the kick, all members of the other team must continue to remain 500 mm away from the ball. Once the kick is taken, the next episode begins.

If the appropriate team has not taken the kick 10 seconds after the command indicating the kick is sent, the referee calls for another stop, followed by a *neutral start*; when the neutral start is called, active gameplay starts immediately.

This structure means that any point in a game naturally falls into one of a few states, and that transitions between these states occur in a predictable fashion: the game might be actively running, or it might be stopped, or it might be waiting for a kick to be taken. In each of these cases, what the robots are doing and what the referee is looking for are qualitatively different from what it is in the other cases. This leads into the idea of representing the game as a hybrid automaton, which is discussed in Section 6.1.

Name	Description
`HALT`	all robots must stop moving immediately; game time stops counting
`STOP`	play temporarily stops to set up a kick; robots must move slowly and stay away from the ball
`INDIRECT(TEAM)`	`TEAM` must take an indirect free kick within 10 seconds
`DIRECT(TEAM)`	`TEAM` must take a direct free kick within 10 seconds
`KICKOFF(TEAM)`	`TEAM` should prepare to take a kickoff
`GOAL(TEAM)`	a goal has just been scored by `TEAM`; also, teams must behave as if the `STOP` command had been given
`READY`	given after `KICKOFF_TEAM`; `TEAM` must take kickoff within 10 seconds
`START`	play starts immediately and any robot may touch the ball

Table 1 The referee commands used in the SSL

3.6 The referee interface: inputs and outputs

The goal of an autoref is to algorithmically replace the human referees. In order to produce a system to achieve that goal, it helps to understand the external interface that an autoref — or, in some sense, a human referee operating a refbox — presents to the other programs on the network.

Viewed from the outside, the autoref receives updates from SSL-Vision in the same way that the teams themselves do, interprets them according to the rules of the game, and outputs commands back to the teams at the appropriate times.

Accordingly, sixty times per second, the autoref receives as input the values in Table 2, which are collectively referred to as a *world state*; we denote by $\mathcal{W}$ the set of all possible world states.

The time is given in seconds since the Unix epoch, and all physical coordinates are given in millimeters, in a coordinate system with its origin at the center of the field, positive x-axis pointing toward a goal, and positive y-axis pointing 90° counterclockwise of the positive x-axis.

As output, an autoref provides two forms of primitives, corresponding to the actions of the human referees: referee commands and ball reset positions. The referee commands are the same as those transmitted by the refbox used by the human referees. The ball reset positions are the replacement for a human referee's ability to physically place the ball on the field: whenever a human referee would be placing the ball somewhere on the field, the autoref outputs a position, and assumes that some

Name	Description
t	the current time
n	the number of robots on the field
T^i	team of the i^{th} robot (`BLUE` or `YELLOW`)
$\mathbf{p}_r^i$	position of the i^{th} robot
$\mathbf{p}_b$	position of the ball

Table 2 The values used as input by the autoref

means exists for the commands to be interpreted and carried out. This means may consist of, for example, a human watching the output of the autoref, a specialized robot, or, with the cooperation of the teams, the players themselves.

The definition of this interface allows an autoref system like the one we describe here to interact with other structurally similar games, not only the SSL. With the appropriate vision input, translation of commands and coordinates, the system could conceivably be used to referee, for example, a game of the MSL instead, using the SSL rules. Of course, the autoref relies upon global ground truth data on the positions of the robots and ball, which would not be generally available in the MSL without additional equipment.

4 Rules of the RoboCup SSL

The rules of the SSL, as described by the technical committee's official rules document (SSL Technical Committee 2015), fall into a few main categories, some of which are easier than others to handle with the autoref.

4.1 Time-based rules

The conceptually simplest and easiest-to-enforce rules are those relating to game time. Each half of the game lasts ten minutes, with a five-minute half time in between. Time starts counting when the `READY` command is sent to teams at the beginning of a half. Teams may also call for a timeout while play is stopped, which halts the countdown of time and allows team members to handle robots and use their computers; per game, each team may take up to four timeouts with a total duration of up to five minutes.

4.2 Robot-based rules

There are a number of rules that are triggered by the interactions between robots themselves. Vision of the robots is lost relatively rarely compared to that of the ball, which helps make these rules easier to check, but this is countered by an element of subjectivity in the first of these rules as listed below.

- If a robot is found guilty of "unsporting behaviour" or "serious and violent contact," (typically, colliding with a slower-moving opponent) an indirect free kick is awarded to the opposing team from the point where it happened.
- If a robot touches the opponent goalie with the point of contact inside the goalie's team's defense area, an indirect free kick is awarded to the opposing team from the nearest legal free kick point.

4.3 Ball-based rules

Finally, some rules are based on the behavior of the ball, or on the interaction between the ball and the robots; there are several limitations on how robots may interact with the ball. Although these rules are straightforward in description, checking them is complicated by the fact that vision of the ball is often lost while it is near a robot, just when these rules might need to be be invoked, and that chip kicks interfere with knowledge of the true position of the ball (see Section 6.6). These rules tend to be the most relevant in real games, so we have focused our efforts here.

- When the ball exits the field without entering a goal, a free kick is awarded to the team which touched the ball less recently. When the exit point is on a touch line (a side of the field not containing a goal), an indirect free kick is awarded from the point 100 mm from the touch line that is nearest to the exit point. When the exit point is on a goal line (a side of the field containing a goal), a direct free kick is awarded; depending on whether the defending or attacking team touched the ball more recently, the kick is taken from the point 100 mm from the nearest touch line and 100 mm or 500 mm, respectively, from the goal line that was crossed.
- When the ball enters one of the goals, a goal is awarded to the appropriate team, the ball is placed in the center of the field, and the team that was scored on takes a kickoff.
- If the ball is kicked into the air (above the maximum height of a robot) and enters the goal without touching a teammate or transitioning from bouncing to rolling, an indirect free kick is awarded to the opposing team from the point where the kick was taken.
- If a robot kicks the ball at over 8 m/s, an indirect free kick is awarded to the opposing team from the point where the kick was taken.

- If a robot drives more than 1000 mm with the ball on its dribbler the whole time, an indirect free kick is awarded to the opposing team.
- If a robot is the first to touch the ball after it takes a free kick (a "double touch"), an indirect free kick is awarded to the opposing team from where the touch occurred.
- If a robot touches the ball while partially inside its own team's defense area, a yellow card is given to its team; if it touches the ball while entirely inside the defense area, a penalty kick is awarded to the opposing team.

A team may also be shown yellow or red cards for a variety of other offenses, such as if it "is guilty of unsporting behaviour," "is guilty of serious and violent contact," "persistently infringes the Laws of the Game," or several others.

Red and yellow cards both decrease by one the number of robots that the receiving team is allowed to have on the field; a red card remains in effect for the remainder of the game, while each yellow card remains in effect for two minutes of game time. (Multiple yellow cards may be in effect for a team at the same time; their effects are cumulative, and each one expires two minutes after it is issued.)

5 Data acquisition

There is a great deal of technical machinery that is needed to bring the data about the physical robots in a game to the software. Although this machinery is not part of the autoref itself, understanding it is helpful for providing context for how the whole SSL ecosystem fits together. In this section, we describe the components that allow an autoref — and game-playing teams in the league — to follow along with the state of the world, as they are a critical part of the flow of information through an SSL system.

5.1 SSL-Vision

In this section, we will give a brief description of how SSL-Vision computes robot and ball positions based on camera input (Zickler et al 2009; Bruce and Veloso 2003; Bruce et al 2000). SSL-Vision processes data from four overhead cameras, each operating at 60Hz, and transmits data from every frame to the teams as soon as it is available.

One part of the setup of SSL-Vision is geometry calibration. Since it would be impractical to require that the cameras be precisely placed in particular positions relative to the field, it is necessary to place them approximately first and determine their positions after the fact. For each camera, a human user must first input:

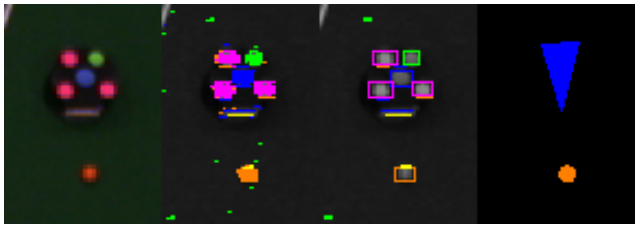
- the height of the camera above the ground;
- the image coordinates of several points on the field with known physical coordinates, which are typically the intersections of field markings; and
- descriptions of the field markings (described as line segments or arcs) within the field of view of the camera.

From the first two of these inputs, SSL-Vision constructs an initial guess at the transformation between field and image coordinates by finding the camera parameters that minimize the sum of squared errors between the given image coordinates and the projections of the world coordinates. Next, it alternates between (a) applying edge detection to the image to extract points which may be on the field markings and (b) refining the guess, assuming that the points are actually on the markings.

The edge detection is performed only within a neighborhood of the projected coordinates of the described field markings, and each detected point is associated with the edge in whose neighborhood it was found. The refinement is then done by adding another term to the error function for each detected point, consisting of the minimum squared error between the detected point and the projection of any point on the marking.

The other part of setup is color calibration. For each camera and each relevant color, a human uses a graphical interface to select a set of bins in YUV color space that should be classified as that color when seen by that camera. The relevant colors are orange, for the ball, along with blue, yellow, pink, and green, for the tops of the robots. SSL-Vision also allows classification of a separate green for the field surface and white for the field lines, but those colors are not currently used during the detection process.

After geometry and color calibration, SSL-Vision is ready to begin working. Each frame received from a camera goes through three main stages of processing: color thresholding, blob detection, and pattern detection. In color thresholding, each pixel is classified according to the bins defined in the color calibration step. In blob detection, contiguous regions ("blobs") of pixels that have been classified as the same color are identified, and their bounding boxes are computed. At this point, any sufficiently large blobs of orange pixels are identified as balls. Finally, in pattern detection, SSL-Vision finds nearby blobs that are arranged in a way that matches the shape of the standard robot pattern, and combines the positions of the blobs to compute an overall position and orientation of the detected pattern. The "butterfly"



(a) (b) (c) (d)

Fig. 5 An example of the stages of SSL-Vision processing. From left to right: (a) a raw image from a camera, (b) the result of per-pixel color thresholding, (c) the result of blob detection, and (d) the result of pattern detection (symbolically representing the computed coordinates of the ball and the coordinates and orientation of the robot)

shape of the pattern was chosen to minimize the final position error after combining the detected blobs.

See Figure 5 for a visual demonstration of the stages of processing.

5.2 State filtering

Although the inputs to the autoref from the physical game field consist solely of time and the positions of game objects, certain derived values are useful; namely, velocities of the objects and estimates of their positions with reduced error, based on the history of observations. Additionally, SSL-Vision reports data from each camera independently; cameras may be out of sync, or multiple cameras may be able to see the same object, and it is preferable to have a fused view where each update includes new information from every camera and multiple views of an object are combined.

To provide this additional information, we apply a discrete-time extended Kalman filter (Simon 2006, p. 407) to the raw SSL-Vision values. A (plain) Kalman filter takes in a sequence of noisy observations of a system whose state vector evolves over time with known linear dynamics, and with a linear mapping from states to observations (along with additive noise). It combines the observations to produce estimates of the state with lower error than the individual observations. Essentially, it functions by performing Bayesian updates on a Gaussian belief state according to the dynamics and observations.

An *extended* Kalman filter generalizes a Kalman filter to use nonlinear dynamics by linearizing the dynamics around the current state; *discrete-time* refers to the property that the dynamics are a discretization of some underlying continuous-time dynamics.

The dynamics of the filter used by our team and our current autoref implementation take into account factors specific to the SSL, including the commands sent to the robots, when available, and collisions between objects. We denote the final derived velocities of the ball and i^{th} robot as $\mathbf{v}_b$ and $\mathbf{v}_r^i$, respectively.

6 Autoref

In this section, we discuss the autoref itself. First, we describe a formalization, as a hybrid automaton, of the structure of an SSL game, and hence of any referee's internal model of the game, and then move on to our implementation of a means of updating the model according to external input.

We formalized the structure of an SSL game by taking the natural-language rules document of the SSL and creating a hybrid automaton that describes the structure of the game. A hybrid automaton is a means of representing a system which contains both discrete and continuous elements that can change over time. We give a brief overview of the hybrid automaton formalism in general (Alur et al 1993), and then of the automaton which specifically describes a game in the RoboCup SSL.

6.1 Hybrid automaton formalism

In its most general form, a hybrid automaton consists of the following components (Henzinger 2000):

- V_D: A finite set of real-valued *variables*. A *valuation* is a map from each variable to a value, and Σ_D is the set of all valuations.
- Q: A finite set of *locations*.
- μ_1: A map from Q to sets of *activities*, which are C^∞ functions from $\mathbb{R}^{\geq 0}$, the set of nonnegative real numbers, to Σ_D.
- μ_2: A map from Q to *exception sets*, which are subsets of Σ_D.
- μ_3: A map from Q^2 to *transition relations*, which are subsets of Σ_D^2. We describe $\mu_3(q_1, q_2)$ as the transition from q_1 to q_2.

For an interval I in $\mathbb{R}^{\geq 0}$, denote its left and right endpoints by l_I and r_I respectively. An interval may be either closed or open at each of its endpoints.

A *trajectory* of a hybrid system is a sequence of tuples $(\sigma_i, \ell_i, I_i, f_i, \sigma_i')$ such that

- For every $i \geq 0$, ℓ_i is a location, σ_i is a valuation, and $(\ell_{i+1}, \sigma_{i+1})$ is a successor of (ℓ_i', σ_i'); that is, $(\sigma_i', \sigma_{i+1}) \in \mu_3(\ell_i, \ell_{i+1})$.
- The I_i are intervals that partition $\mathbb{R}^{\geq 0}$ and are in increasing order by index.
- For every $i \geq 0$, (a) f_i is in $\mu_1(\ell_i)$, (b) $f_i(0) = \sigma_i$, (c) $f_i(r_{I_i} - l_{I_i}) = \sigma_i'$, and (d) for all $t \in I_i$, $f_i(t - l_{I_i}) \notin \mu_2(\ell_i)$.

In effect, a trajectory describes a possible evolution of the automaton over time, where the state may evolve continuously according to the trajectories or discretely according to the transition relations. The exception sets define conditions which force the automaton to undergo a transition out of a location. A common scheme, which we will use here, is to express the sets of activities as the solution sets of differential equations. Additionally, an automaton may effectively include discrete variables if those variables are constant in all allowed activities and change only in transitions.

Hybrid automata are mostly used in the context of formal verification, and so authors are typically concerned with characterizing the set of all possible trajectories of an automaton over time. Instead, we think of and describe an automaton as a machine which follows a specific trajectory in any given instantiation.

Accordingly, we may replace the transition relations with *transition functions*, so that the range of μ_3 is instead the partial functions $\Sigma_D \to \Sigma_D$. Then we are concerned with trajectories such that for each i, $\sigma_{i+1} = \mu_3(\ell_i, \ell_{i+1})(\sigma'_i)$.

We also define the exception sets implicitly by associating a guard condition (a predicate over valuations) with each transition; the exception set for each location will then consist of all valuations which satisfy the guard condition for some transition, and the domain of each transition function will be the set of valuations satisfying the associated guard condition. This forces the automaton to transition as soon as the current valuation satisfies the guard condition for some transition.

6.2 SSL automaton

For the automaton describing an SSL game specifically, the locations, which are described in Table 3, capture the idea that, roughly, the game is either running or stopped, and each of those entails a distinct set of conditions that are relevant to refereeing. The auxiliary variables required to represent the state of the game are described in Table 6, and the differential equations describing the continuous evolution of the variables is described in Table 4. One of the variables, the current stage, describes what overall part of the game is currently happening: either one of the halves, halftime, or the time before or after the whole game; these values are described in Table 5.

We describe the occurrence of a transition in the automaton as an *event*; the continuous evolution of the automaton is interrupted whenever an event occurs. For the implementation, events are triggered by detector objects that examine the current world state and check whether any relevant rule applies to the current situation.

We also define some notation and functions operating on the values of these variables, for convenience.

- `team(R)`: the team of robot `R`
- `stage-end-cmd(s)`: the command that is given to end stage `s` (`KICKOFF(T)` for halftime, `HALT` for game halves)
- `stage-length(s)`: the length in seconds of stage `s`
- `next-stage(s)`: the stage that comes after stage `s`
- `!T`: the opponents of team `T`

In this presentation, most of the predicates are described in plain text, with the details of how to receive inputs and evaluate them left abstracted away. It should be possible to incorporate a more detailed model of their computation into the automaton without drastic changes, by adding some additional variables and self-loops. Similarly, the change over time of the variables describing the world state can be considered to evolve according to complex equations which are part of the automaton but unspecified. It may also make sense to actually model the event detection as a separate automaton; the hybrid automaton model allows for the composition of two automata, which can communicate by emitting and receiving events, and the generation of the primitive events can be thought of as coming from another unspecified automaton.

Figure 6 shows a simplified representation of the automaton, with only the set of locations and which pairs have edges between them; each edge there corresponds to multiple possible transitions, with the transitions between each pair of locations given a high-level description of what kinds of rules form it. The details of the transitions are specified in the appendix. For simplicity, we have omitted the detailed rules for penalty kicks and penalty shootouts, as well as the calculations for kick reset positions. (A detailed example of the position calculation is given in Algorithm 2 in Section 6.5.) In each table, the first column represents the preconditions on the world state for the transition to occur, the second represents the referee command component of the action that is transmitted when the transition occurs, and the third column represents the changes made to the internal state variables.

The overall locations correspond to those in mentioned in Section 3.5. When the game is in the GAME_ON location, the ball is in play and the robots are actively trying to score goals; during GAME_OFF, the ball is being placed for the next restart of play; during SETUP, the ball has been placed and the robots may set up for a kick.

We have described this automaton as abstractly representing the structure and state of a game, as defined

Name	Domain	Description
`time`	$\mathbb{R}$ (seconds)	the current time
`stage-end`	$\mathbb{R}$ (seconds)	the time at which the current stage will end
`timeout-time(T)`	$\mathbb{R}_{\geq 0}$ (seconds)	remaining timeout time for team `T`
`timeout-num(T)`	$\mathbb{N}$	the number of remaining times that team `T` may call a timeout
`reset-position`	$\mathbb{R}^2$ (meters)	the position where the ball should be placed for the next kick
`kick-deadline`	$\mathbb{R}$ (seconds)	the time at which the game will be reset to a forced start if the current kick is not taken yet
`call`	referee commands (Table 1)	the command that should be transmitted next (used while the game is stopped)
`stage`	game stages (Table 5)	the current coarse stage of the game
`touch-team`	{blue, yellow}	the last team to touch the ball while the game was running
`kick-team`	{blue, yellow}	the team currently permitted to kick the ball
`goal-allowed`	{true, false}	whether a direct goal is currently allowed (true except immediately after indirect free kicks)
`kicker`	{robots on the field} $\cup$ {*null*}	the robot which just took a free kick
`r-cards(T)`	$\mathbb{N}$	number of red cards for team `T`
`y-times(T)`	lists of elements of $R_{\geq 0}$	time remaining for each yellow card for team `T`

Table 6 The auxiliary variables of the automaton representing an SSL game

by the rulebook, but there is a direct correspondence to what human referees and autorefs must do while they observe the game. They are also essentially modeling the same automaton: they must use their observations to estimate when the true state of the game ought to change, so that they can issue the appropriate commands.

6.3 Automaton evolution and the rules

In this section, we discuss some examples of how the changes in game state combine with the rules of the game to produce the transitions that the automaton encodes.

Imagine that the game is actively being played, but then the ball goes out after being touched by the blue team; play must stop so that the referee may retrieve and place the ball, then signal the next kick. However, suppose the team has a software problem and does not take the kick before 10 seconds have passed, so the referee stops the game again and then calls for a neutral restart. In order, the individual components of the changes in the automaton state are as follows:

- The location is GAME_ON.
- The blue team touches the ball. The touch event sets the `touch-team` variable to `BLUE`.
- The ball goes out. The `STOP` command is transmitted, which tells teams that the game is stopped and that they should prepare for some kick to come next; internally, the location changes to GAME_OFF, `call` is set to `INDIRECT`, and `kick-team` is set to `YELLOW`.
- Once the ball is positioned and the robots settle, the `INDIRECT(YELLOW)` command is transmitted, which

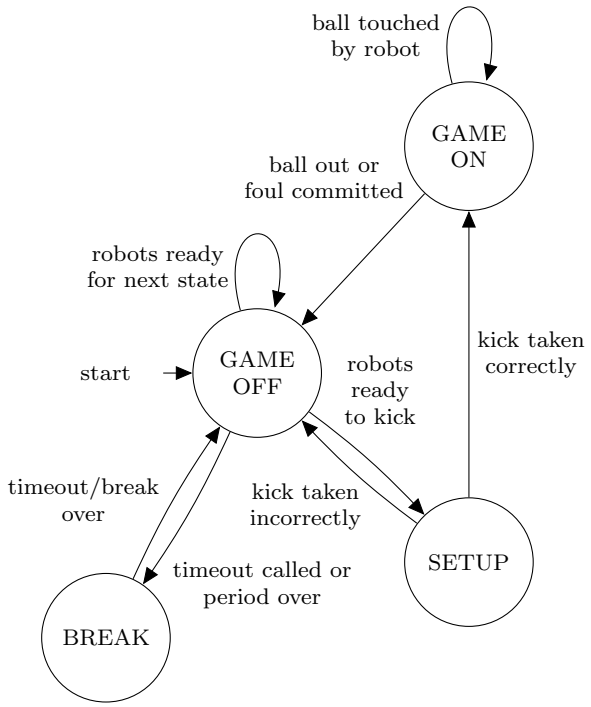


Fig. 6 A simplified representation of the automaton representing a game of the SSL

Location	Description
GAME_ON	The game is actively being played. This location typically ends when a robot commits a foul or the ball leaves the field.
GAME_OFF	The game has been stopped for one of the preceding reasons, and the robots must not come near the ball while the referee is positioning it. This location ends once the robots are ready to kick, which is usually taken to be when they stop moving or after some time has passed.
SETUP	The command to take a kick has been given, but the kick has not been taken yet. This location ends once the kick is taken, or too much time has passed without a kick.
BREAK	One team has called a timeout or the game is in a break between periods. This location only ends if the team ends the timeout or time in the break period runs out.

Table 3 The high-level automaton locations of an SSL game

Locations	Equations
all	$\frac{d}{dt}$`time` $= 1$
BREAK	$\frac{d}{dt}$`timeout-time(timeout-team)` $= -1$
all but BREAK	$\frac{d}{dt}x = -1$ for all $x \in$ `y-times(BLUE)` $\cup$ `y-times(YELLOW)`

Table 4 The differential equations for the continuous variables of an SSL game; see Table 6 for descriptions of the variables. Most of the continuous evolution of the game is under the direct control of the team agents, rather than the referee, so these equations are minimal. By a slight abuse of notation, the third equation indicates that all the elements of `y-times(BLUE)` and `y-times(YELLOW)` have derivative -1

Name	Description
`PRE-FIRST`	the `READY` command for the first kickoff of the first half has hasn't been given yet
`FIRST`	the initial `READY` command for the first half has been given, and the first half hasn't ended
`HALFTIME`	the first half has ended and the robots are halted
`PRE-SECOND`	the first half has ended and the initial `READY` command of the second half hasn't been given yet
`SECOND`	like `FIRST`, but for the second half
`POST`	the second half has ended

Table 5 The coarse stages of an SSL game

tells the yellow team that it is to take a kick within 10 seconds. Accordingly, the location changes to SETUP and `kick-deadline` is set to `time + 10`.
- Once `time` becomes greater than `kick-deadline` (i.e., 10 seconds pass and the team does not take the kick), the `STOP` command is transmitted, since there is about to be a new restart of the game; the location changes back to GAME_OFF and `call` is set to `START`.
- Once the robots settle again, the `START` command is transmitted to indicate that gameplay has resumed and either team may touch the ball, and the location changes to GAME_ON.

As another example, consider the sequence of events that occurs when a goal is scored by the yellow team. According to the definition of the commands as given by the SSL, after the game is stopped there must be a `KICKOFF` command for the appropriate team, which is then followed by a `READY` command, and only then is the team allowed to actually kick the ball — as opposed to a free kick, where only a single command is given after the game is stopped.

- The location is GAME_ON.
- The ball enters the blue goal, so the `GOAL(YELLOW)` command is transmitted to indicate to the teams that the yellow team has scored. The game must now stop to set up for a kickoff by the blue team, so the location changes to GAME_OFF, `call` is set to `KICKOFF`, and `kick-team` is set to `BLUE`.
- Once the ball is positioned and the robots settle, the transition from GAME_OFF to itself is taken; the `KICKOFF(BLUE)` command is transmitted, but the location remains equal to GAME_OFF.
- Once the robots settle again, now knowing that it will be a kickoff for the blue team, the location becomes SETUP and the `READY` command is transmitted, indicating that the blue team may take the kick now.
- Once the blue team takes the kick, gameplay has started again, so the location becomes GAME_ON.

Figure 7 graphically demonstrates the sequence of transitions that occurs at the beginning of a typical game, with a kickoff given, a brief period of play, and then a ball exit leading to an indirect free kick.

Finally, we briefly discuss the use of the `kicker` and `goal-allowed` variables, which are used for two intertwined rules. First, if a robot takes a free kick and is the first robot to touch the ball afterward, play stops and the other team receives an indirect free kick. Second, a goal may not be scored directly from an indirect free kick; that is, a goal is not allowed if the current episode was started by an indirect free kick and the last robot

to touch the ball is the robot that took the kick. (If the ball enters a goal, it is treated as simply having left the field over the goal line.) Accordingly, when an indirect free kick is signaled, `goal-allowed` is set to false; for any other kick, it is set to true. When a kick is actually taken, `kicker` is set to the robot which took it. As long as the ball is in play and no robots touch it, neither variable changes. When a robot touches the ball, what happens next depends on whether that robot is the same as `kicker`. If it is, this is a double touch and the game stops. Otherwise, `kicker` is set to *null* (i.e., some sentinel value that is equal to no robot), since a double touch is no longer possible. Simultaneously, `goal-allowed` is set to true, since a second touch means a goal is now allowed.

6.4 Event loop algorithm

We implemented a program to approximately track the evolution of the automaton described above in terms of a set of separate *event detectors*, each of which receives the sequence of world states and outputs an object describing how the state of the automaton should be updated. The algorithm implements a sort of discrete approximation of the true continuous-time automaton, since it can only receive discrete updates of the world state. It operates at the full frame rate of SSL-Vision

The architecture described here is based on that of CMCast (Veloso et al 2008); however, we emphasize that the overall algorithm can implement a general hybrid automaton, with a suitable set of event detectors plugged in.

In this kind of architecture, an event is any of the conditions associated with an edge in the automaton. The events we have currently implemented are:

- the ball's speed goes above the maximum limit defined in the rules,
- the ball enters a goal,
- the ball exits the field without entering a goal,
- the robots are ready for a free kick to be taken,
- a robot touches the ball,
- a robot takes a free kick,
- a robot dribbles the ball over a linear distance of greater than 1000 mm,
- a team is taking too long to take a free kick,
- progress gets stuck during play, and
- the end of the first or second half is reached.

Each time a new world state is available, the autoref algorithm checks whether any event has occurred; if so, it updates its state variables and may issue a command accordingly.

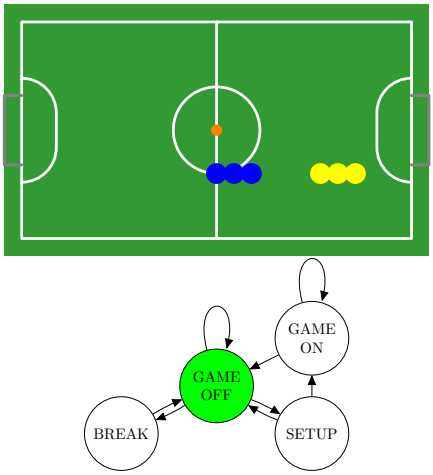
The state and values of the auxiliary variables of the automaton are described by $v \in \mathcal{V}$. We also make use of a set of *action* objects, each representing an externally visible action that the program can take. The contents of $\mathcal{A}$ may vary depending on the application. For the RoboCup SSL, any particular action consists of the outputs described in Section 3.6: signaling that the ball should be positioned at a particular point or transmitting a new referee command to teams (and possibly both or neither).

In general, an event may conceptually be thought of as depending on the full history of world states; however, we expect that events in practice will not need to truly operate on the whole history, but can instead rely on a small fixed set of variables calculated from each new world state and the previous values of those variables. For the purpose of fitting within the hybrid automaton framework, it is also required to limit the system to a fixed number of variables. Therefore, with each event e we associate a set $\mathcal{S}_e$ which contains possible "summaries" of the world state as relevant to the event. Then the event can be defined as a function of type $\mathcal{W} \times \mathcal{S}_e \times \mathcal{V} \to (\mathcal{S}_e \times \mathcal{V} \times \mathcal{A}) \cup \{null\}$: each time a new world state is received, the function is provided with that state, its own summary of previous world states, and the current internal variables, and it returns either *null*, indicating that the event did not fire, or the new summary, variables, and an action to take.

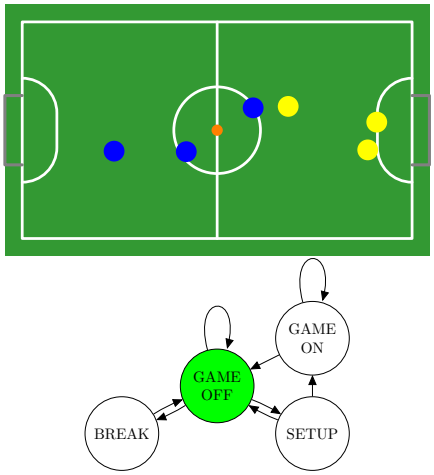
The main action selection algorithm is given in Algorithm 1, where E denotes the list of relevant events and s_e denotes the summary for an event $e \in E$. This algorithm first updates the variables according to the differential equations of the current location (denoted by the call to ContinuousUpdate), then handles the discrete transitions. It finds the first event that is firing, executes the action, and updates its variables; then it returns the beginning of the list and repeats until no events fire. This allows multiple transitions to occur due to the same input, which is occasionally necessary, as discussed by Mosterman (1999). Since the continuously updated global variables for the autoref are particularly simple, we will not further discuss the continuous update.

6.5 Event detector implementations

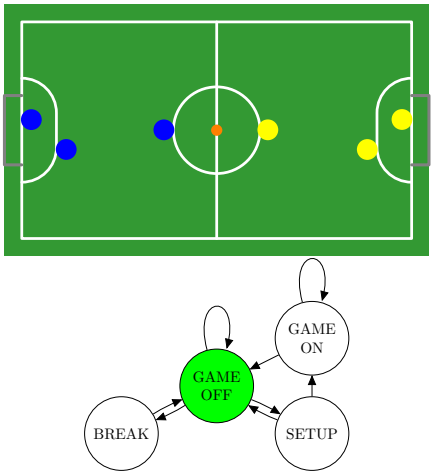
Now we provide concrete examples of the implementations of events to demonstrate the detection algorithms and how the event loop corresponds to the hybrid automaton. Note that for this specific automaton, the only variables which update continuously are the current time and, during a team's timeout, the amount of timeout time remaining to that team.



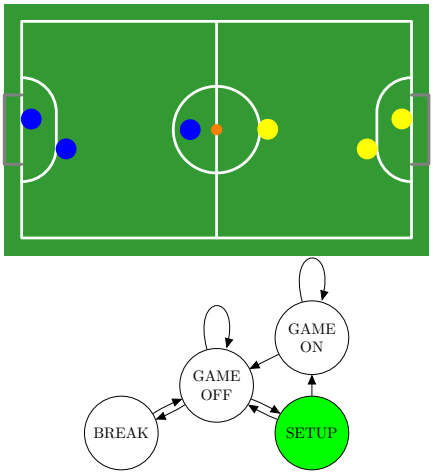
(a) The robots are in their starting positions when the autoref first begins execution and transmits the `STOP` command



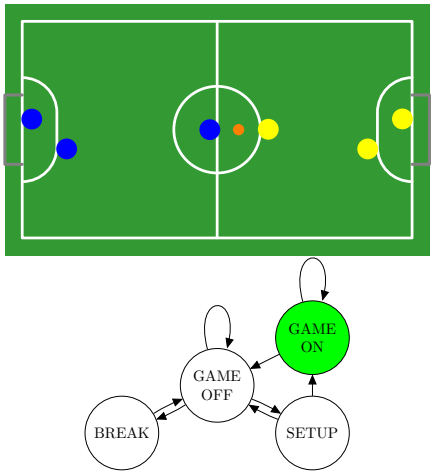
(b) The robots have begun moving to new positions in response to the `STOP` command



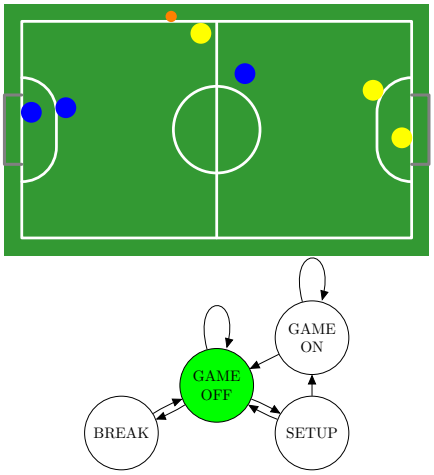
(c) The robots have settled, so the autoref now transmits `KICKOFF` and waits for them to settle again



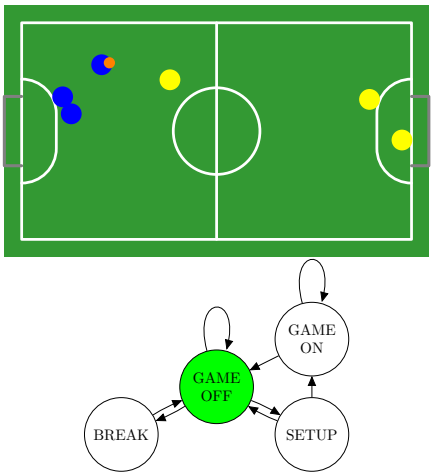
(d) The robots have settled again; the autoref now transmits `READY`, so the blue team may take the kick



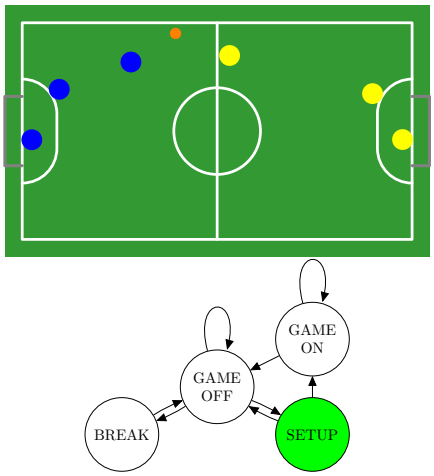
(e) The blue robot has just taken the kick, so the location becomes GAME_ON



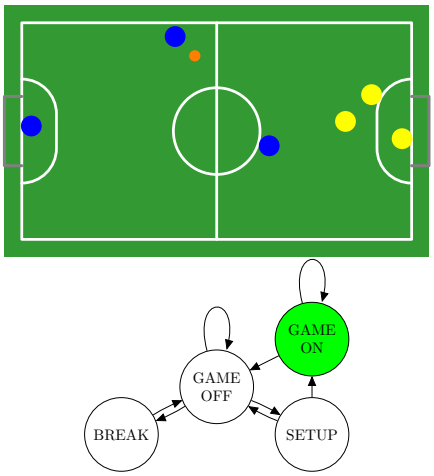
(f) The ball has gone out of the field; the location becomes GAME_OFF so that the ball may be positioned for a free kick



(g) The referee robot (here, the blue goalie) pushes the ball to its reset location



(h) The robots have settled, so the blue team is free to take the kick



(i) The blue team has taken the kick and the location returns to GAME_ON again

Fig. 7 A demonstration of the evolution of the autoref over time in response to a typical sequence of motions of the robots and ball. Each subfigure shows the field at some moment, with the corresponding automaton location highlighted below

Algorithm 1 The overall event-handling loop algorithm

w: world state
v: values of the autoref variables

```
1:  function RunEvents(w, v)
2:      ContinuousUpdate(w, v)
3:      fired ← true
4:      while fired = true do
5:          fired ← false
6:          for e ∈ E do
7:              ret ← e(w, s_e, v)
8:              if ret ≠ null then
9:                  v, s_e, a ← ret
10:                 fired ← true
11:                 execute a
12:                 break
13:             end if
14:         end for
15:     end while
16: end function
```

In the pseudocode listings below, v and a refer to elements of $\mathcal{V}$ and $\mathcal{A}$. Construction of v and a are expressed by assignments to subscripted components of them, corresponding to the variables described in Table 6 and actions described in Section 3.6:

- v_{call}: the command that should be transmitted next.
- v_{loc}: the new location of the referee automaton.
- v_{last_touch}: the last team to touch the ball.
- a_{cmd}: the command to be sent to the teams immediately.
- a_{reset}: the point at which to place the ball for the next kick.

Additionally, assignment to s is used in a similar fashion to indicate the current event-specific state summary, a member of $\mathcal{S}_e$.

Each event may also refer to the positions of the objects, $\mathbf{p}_b$ and $\mathbf{p}_r^i$, and their velocities, $\mathbf{v}_b$ and $\mathbf{v}_r^i$.

6.5.1 Ball exit

Most episodes end with the ball exiting the field, making it one of the most common events in a game. Algorithm 2 shows a simplified version of the code used to detect when the ball has left the field and determine the appropriate response.

Lines 2–11 determine whether the ball exit event should be considered to be happening in the current frame. It uses s_{last} as a counter, so that it only makes the call if the observed position of the ball is outside the field for 3 consecutive frames, in order to reduce the incidence of false positives due to incorrect data. The remaining lines determine the appropriate response if the event is happening. Lines 13–14 indicate that, in any case, the game should be stopped; lines 15–19 assign the kick to the appropriate team. Finally, lines 21–30 compute where the ball should be placed for the subsequent free kick, according to whether the ball passed over the touch line and should be a throw-in, or passed over the goal line and, depending on which team touched it last, should be a goal kick or corner kick.

Algorithm 2 The algorithm for determining whether a the ball has left the field and how to respond if so. CrossBoundaryPoint returns the intersection of the line segment between its two arguments with the boundary of the field. Although w is symbolically passed as a parameter to avoid clutter, the algorithm uses the variables defined in Table 2, which are considered to be part of w

w: world state
s_0: initial value of the world summary for this event
v_0: initial values of the autoref variables
s: updated value of the world summary for this event
v: updated values of the autoref variables
a: action to take if this event fires

```
1:  function BallExitEvent(w, s_0, v_0)
2:      s ← copy(s_0)
3:      v ← copy(v_0)
4:      a ← empty action
5:      if InsideField(p_b) then
6:          s_last ← p_b
7:          s_count ← 0
8:          return null
9:      else
10:         s_count ← s_count + 1
11:     end if
12:     if s_count < 3 then
13:         return null
14:     end if
15:
16:     a_cmd ← STOP
17:     v_loc ← GAME_OFF
18:     if v_last_touch = YELLOW then
19:         v_call ← INDIRECT(BLUE)
20:     else
21:         v_call ← INDIRECT(YELLOW)
22:     end if
23:
24:     c_x, c_y ← CrossBoundaryPoint(s_last, p_b)
25:     if OnTouchLine(c) then
26:         a_reset ← ⟨c_x, c_y − 100 · sgn(c_y)⟩
27:     else
28:         if v_touch_team = DefenseTeam(sgn(c_x)) then
29:             a_reset ← ⟨sgn(c_x)·(L−100), sgn(c_y)·(W−100)⟩
30:         else
31:             a_reset ← ⟨sgn(c_x)·(L−500), sgn(c_y)·(W−100)⟩
32:         end if
33:     end if
34:     return s, v, a
35: end function
```

6.5.2 Ball touching

Detecting when a robot has touched the ball is crucial to accurate refereeing, and is one of the most difficult individual events to detect.

We started off with the touch detector that was already present in the CMDragons codebase, which we will call the *accel* algorithm. Each frame, it estimates the acceleration of the ball by computing the difference between the last ball position and the average of the current and second-to-last positions. When this acceleration is above a certain threshold, the frame is treated as a potential collision. If any robot is within a certain distance of the ball, the closest robot is considered to have just touched the ball.

We implemented two additional algorithms for detection, *line* and *backtrack*, to improve the overall performance of touch detection. We found that all of these algorithms have fairly low false positive rates, so we combine them simply by reporting a touch when any individual algorithm does so.

Both of the new algorithms are based on, for each robot, the history of positions of the ball relative to that robot; that is, $\mathbf{p}_b - \mathbf{p}_r^i$ for each i for every frame. Let $\mathbf{d}_n$ represent this difference in the n^{th} world state before the current one (dropping the i index, since each robot is processed independently and identically).

The *line* algorithm checks whether the recent history of relative positions match a two-part broken line, with the intersection point near a robot at the appropriate time; the idea is that the trajectory of the ball consists mostly of straight segments, separated by times when it collides with a robot. Let R be the radius of the robot and r be the radius of the ball. Then, according to this algorithm, a robot has touched the ball when the following conditions are true:

- $\mathbf{d}_1$ is "between" $\mathbf{d}_0$ and $\mathbf{d}_2$, meaning $\left|\mathbf{d}_1 - \frac{\mathbf{d}_0+\mathbf{d}_2}{2}\right| < .1 \cdot |\mathbf{d}_0 - \mathbf{d}_2|$;
- $\mathbf{d}_4$ is between $\mathbf{d}_3$ and $\mathbf{d}_5$;
- the intersection of the lines $\overline{\mathbf{d}_0\mathbf{d}_2}$ and $\overline{\mathbf{d}_3\mathbf{d}_5}$ has magnitude less than $R + r + 30\,\text{mm}$;
- $\frac{\mathbf{d}_2-\mathbf{d}_0}{|\mathbf{d}_2-\mathbf{d}_0|} \cdot \frac{\mathbf{d}_5-\mathbf{d}_3}{|\mathbf{d}_5-\mathbf{d}_3|} < .99$ (which checks, roughly, whether the points are not all part of the same line).

The *backtracking* algorithm checks whether tracing the current trajectories of the ball and each robot backwards in time results in a projected collision. The idea is that the changes from straight-line trajectories are usually caused by collisions with robots, so if the most recent trajectory looks like it came from inside a robot, it must be the result of a collision.

The algorithm starts by examining $\mathbf{d}_0$, $\mathbf{d}_1$, $\mathbf{d}_2$, and $\mathbf{d}_3$; if the line segment between any consecutive two of them contains a point with magnitude less than $R + r$, then it aborts. This is an attempt to filter out false positives caused by chip kicks passing over robots. If the ball does not appear to have just passed over the robot, the algorithm fits a constant-velocity trajectory to $\mathbf{d}_0$, $\mathbf{d}_1$, $\mathbf{d}_2$, and $\mathbf{d}_3$ using linear least-squares, and extrapolates that trajectory 2 frames further into the past; if either extrapolated vector has a magnitude less than $R + r - 10\,\text{mm}$, the robot is considered to have touched the ball.

6.5.3 Robots settled

Another important event to detect involves deciding when the robots are ready to resume play by taking a kick. Waiting for the robots to settle down is not mentioned in the rules, and so there is no standard description of how to make the decision or, strictly speaking, any need to wait at all. However, this behavior is so common among human referees that we decided that it made sense to emulate it.

After examining the usual behavior of human referees, we decided to base this event on the speed of the robots. Each frame, the algorithm examines the speed of the robots and the ball. If all the speeds are below particular thresholds (possibly different for robots and the ball) continuously for a period of time, then the algorithm decides that the robots are ready. This algorithm is shown in Algorithm 3. The variable s_{start} represents the last time that the robots or ball were moving too fast. Lines 9–16 check whether all the speeds are below the appropriate thresholds; if not, they reset the s_{start} variable to the current time. Lines 17–19 check whether it has been sufficiently long since s_{start} was last reset, i.e., whether the speeds have been low enough for long enough. If execution continues past that point, then the robots are deemed to be ready; the remaining code determines how to respond.

The following code examines v_{call}, which should have been used by a previous event to remember what call should be made next. If it indicates a free kick or the ready signal (for a kickoff or penalty), then a kick can now be taken, so the new automaton location should be SETUP. If v_{call} indicates a neutral restart, that means both teams are free to manipulate the ball and the location is GAME_ON. Finally, if v_{call} indicates a kickoff or a penalty, then there still needs to be a `READY` command sent, so the location is still GAME_OFF, but the next command to be sent is `READY`. Note that s_{start} is reset here, because we are waiting for a new transition period and settling. If it were not reset, then the event would fire again immediately on the next evaluation.

Algorithm 3 The algorithm to check whether the robots are ready to take a kick to restart the game

w: world state
s_0: initial value of the world summary for this event
v_0: initial values of the autoref variables
s: updated value of the world summary for this event
v: updated values of the autoref variables
a: action to take if this event fires

```
1: function KickReadyEvent(w, s_0, v_0)
2:     s ← copy(s_0)
3:     v ← copy(v_0)
4:     a ← empty action
5:     if v_loc ≠ GAME_OFF then
6:         return null;
7:     end if
8:     ready ← true
9:     if |v_b| > V_b then
10:        s_start ← t
11:    end if
12:    for i = 1, ..., n do
13:        if |v_r^i| > V_r then
14:            s_start ← t
15:        end if
16:    end for
17:    if t − s_start < 1 then
18:        return null
19:    end if
20:    a_cmd ← v_call
21:    if v_call = INDIRECT
22:      or v_call = DIRECT
23:      or v_call = READY then
24:        v_loc ← SETUP
25:    else if v_call = START then
26:        v_loc ← GAME_ON
27:    else if v_call = KICKOFF
28:      or v_call = PENALTY then
29:        s_start ← t
30:        v_call ← READY
31:    end if
32:    return s, v, a
33: end function
```

6.6 Obstacles

The domain-specific logic is contained entirely within the detection of individual events, so it is critical that the event detection be reliable. In the RoboCup SSL, the primary obstacles are unreliable data and chip kick detection.

Although SSL-Vision returns precise and complete position data with high reliability, occlusion of the ball and the brittleness of the color calibration scheme mean that robots or the ball can be lost from vision for many frames at a time. This is particularly a problem for judgments which involve contact between robots and the ball (mainly, touching the ball or dribbling it for too great a distance); vision of the ball is often completely lost during those situations, just when we need it the most.

The other main problem relates to chip kicks. Because SSL-Vision returns a ball position assuming that the ball is on the ground, the values it gives are not correct when the ball is in the air; they instead give the intersection of the ground plane with the line containing the camera and the ball. There are methods to detect when this is happening and determine the true trajectory of the ball, but we have yet to come up with an implementation that is sufficiently sensitive to be of use without giving too many false positives. As a result, many judgments based on the position of the ball may be evaluated incorrectly in the presence of chip kicks, especially checking whether the ball has left the field: due to the typical positioning of the cameras above the center of each half of the field, most chip kicks near the edges of the field appear to take the ball outside the field at their peaks.

7 Results

In order to test the abilities of the autoref, we performed two types of evaluation. First, to test the overall stability and rule adherence of a game as judged by the autoref, we set the autoref up to run repeated simulated games, with two copies of the CMDragons team playing, and collected statistics about the results of those games. Second, to test the ability of the individual event detectors to operate on real data, we ran some of the detectors on data from games of RoboCup 2014 and compared the results to the calls made by the human referees at those games.

7.1 Running games

Although not all of the rules of the SSL are currently implemented, the ones that are make up the great majority of rules that come into effect during games in practice. With the currently implemented events, it is possible to run an essentially full game in simulation, with kickoffs, appropriately awarded free kicks for ball exits and some fouls, and checks to ensure that the progress of the game is not delayed by a slow team or stuck ball. As a result, the autoref makes it feasible to run and gather information from a far greater number of rules-compliant games than would be feasible to monitor by hand.

As a simple demonstration of this capability, we ran many simulated games comparing two slightly different versions of our own CMDragons soccer team. In some tests, we enabled or disabled some new feature of the team, so that we could check whether having that feature gave the team an advantage. In others, one version of the team was handicapped, either by having fewer robots

or by having the speed and acceleration of its robots restricted to a fraction of their normal values. Overall, we have run thousands of games, and can run well over one thousand games continuously without supervision.

We found that, when one team is simply handicapped, the other has a noticeable advantage; while this is no surprise, the ability to verify this through experiment is made possible only by the presence of an autoref. The average scores as functions of the degree of handicap are shown in Figure 8; the error bars indicate the standard error of the mean after 100 games. We demonstrated similar results in Zhu et al (2015), but the version of the autoref being used here implements a more complete subset of the rules of the game.

We also ran some tests to determine the effect of software features on performance. One of the features we tested was a zone-based attack strategy we developed and used in the RoboCup tournament last year; the other was a method for a robot to quickly turn while dribbling the ball, which we used for only part of the tournament because it was unreliable on real robots (Veloso et al 2015). The results are shown in Table 7; both features provided an advantage, as we hoped. However, this is one case where the limitations of the simulation environment must be kept in mind: the fast turning behavior worked very well in simulation, but we disabled it in reality because it did not work as well. The zone-based attack seems less likely to be dependent on the specifics of the simulation.

The modularity and common network interfaces of the various components of the SSL ecosystem also mean that the same autoref code is compatible with live games played on the physical robots. Of course, there are other concerns, such as robot batteries needing to be changed and the ball getting stuck outside the view of SSL-Vision. Even more important is the ability to position the ball for a kick. In simulation, we simply command the ball to be at a certain location, and it is there, but if we are to avoid requiring a human to position the ball every time there is a kick, we need some real-world analogy; using a ball-positioning robot is the natural choice.

For technical ease, we chose to implement the ability for the soccer teams themselves to listen to ball-positioning requests and use their own robots to fulfill them. When the ball needs to be placed, the goalie of one of the teams drives out to place the ball on its dribbler and pushes it to the desired location. Although the current positioning procedure is not quite fast or reliable enough for practical use, the essential capabilities are already in place to run full games with little human intervention. We have also developed hardware for a specialized referee robot, which has improved the situation; it has an arm that can completely encircle the ball, holding it solidly in place in front of the robot. The main advantage of such a robot is that, by the rules, player robots may not be designed to surround the ball or remove all of its degrees of freedom, but such capabilities are of great use precisely for the task of quickly and repeatably positioning the ball on the field.

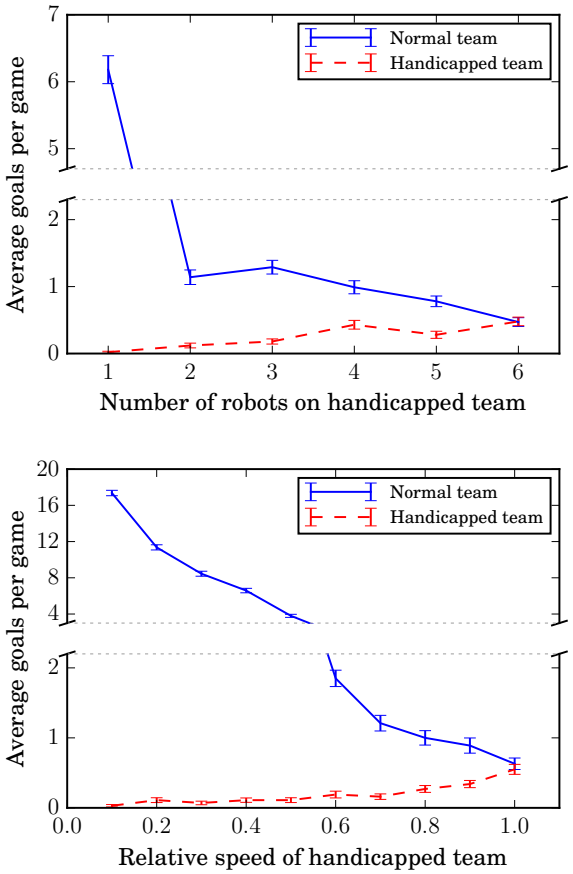


Fig. 8 Comparisons of two versions of our team, one handicapped by having either fewer robots or reduced speed and acceleration. Each data point depicts the mean score and standard error of the mean of the two teams across 100 full games

	Zone-based attack		Turn & dribble	
	Win %	Goals	Win %	Goals
feature on	38.00	0.58 ± 0.05	42.00	0.70 ± 0.05
feature off	12.67	0.18 ± 0.02	10.67	0.22 ± 0.03

Table 7 Comparisons of two versions of our team, differing by the presence or absence of a new feature. "Win %" indicates the percentage of games won by that team, and "Goals" indicates the mean number of goals per game, as well as the standard error of the mean. Each comparison is based on 300 games

7.2 Comparison to human refereeing

To further test the accuracy of the autoref implementation, we compared its performance against the human referees of games from RoboCup 2014. For each of the quarterfinal (QF), semifinal (SF), and final (F) games, we manually reviewed our log files of those games, determined the times that a rule that the autoref can currently detect should have been applied, and recorded whether the referee actually made each call during the game. We then ran the autoref as if it were receiving the log data as real input, and compared the set of calls made by the autoref to the set of calls that were made.

We also noted which calls were made incorrectly by the human referee. In manually reviewing the logs, we have the advantage of being able to stop and examine the input from vision frame by frame, which we believe gives us enough certainty in most cases to tell whether a call was correct, even when our call differs from the referee's. Some cases remained ambiguous, usually when two or more robots obscured the ball while it exited the field; these were counted separately.

7.2.1 Ball exit calls

By far the most common call in a typical game is to stop the game when the ball exits the field and award a free kick to the opponents of the team which touched the ball most recently.

The results of both the manual annotation and the automatic detection by the autoref consist of a list of $\langle time, team \rangle$ pairs. A pair on one list is considered to match one on the other if their times are within 0.3 s of each other; each event can match at most one event on the other list. The results of this comparison are shown in Figure 9. Annotated events which don't match to any detected events are described as *missed*, and *extra* events are the reverse.

There were no calls that were missed, and most of the extra calls made are due to chip kicks near the edge of the field; as discussed in Section 6.6, such kicks can cause the ball to appear as though it is outside the field when it is not. If the ball actually does leave the boundaries of the field during a chip kick, this could result in both a missed call and an extra call, as the call is made well in advance of the true exit, although that did not occur in any of these games. The only other extra calls were two during the final game that resulted from some brief span of spurious data from SSL-Vision, in which the ball rapidly flickered in and out of view and, for a few frames, appeared to be outside the field. (It may have gotten on top of a robot, which would both interfere with detection and, even when it was being detected, caused it to appear further out from the camera than it actually was, just as chip kicks do.)

	QF	SF	F
last touch clear	47	59	52
last touch ambiguous	1	3	6
human referee			
correct	43	55	52
last touch incorrect	4	4	0
autoref			
correct	37	52	43
last touch incorrect	10	7	9
extra	0	0	2
chip extra	3	1	6

Fig. 9 Performance of the autoref for ball exit events as compared to human referees at RoboCup 2014. The *correct* lines refer to calls which were made at an appropriate time, with the subsequent kick awarded to the correct team; *last touch incorrect* refers to a call made at an appropriate time, with the kick going to the wrong team. The *extra* line for the autoref refers to calls made by the autoref that did not result from actual ball exits, except for those caused by chip kicks near the edge of the field, which are counted under *chip extra*

Among the calls that whose presence was correctly identified, the proportion of team misidentifications by the autoref is larger than would be desired for the final version of the program, at about 15% of all ball exit calls. The touch detection algorithm is still incomplete and under development. Many of the incorrect identifications are also due to chip kicks; the ball may appear to pass very near or through a robot, registering as a touch, when in fact it passed over the robot.

We also compared the results of the different combinations of touch detection algorithms; this is shown in Table 8. We show the performance of the previous baseline algorithm (in the first line) with the results of the human refereeing during the actual games (in the last line) and the various combinations of improvements (in between). The new algorithms we developed improved on what was previously available in our codebase, and it was promising to see that combining the different algorithms yielded the greatest improvement overall, though we have yet to equal the performance of a live human referee.

7.2.2 Maximum ball speed calls

The rule stipulating that the ball cannot be kicked above 8 m/s often goes uncalled when it is violated, since it is difficult for a human referee to accurately determine the speed of the ball. By contrast, the ball tracker used by the autoref already has accurate quantitative information about the ball, and so checking for violations of this rule is straightforward.

Algorithm	QF	SF	F
accel	.66	.66	.73
accel + line	.68	.73	.77
accel + backtrack	.79	.85	.83
accel + backtrack + line	.79	.88	.83
human	*.89*	*.93*	*1.00*

Table 8 The fraction of times that the autoref or human referee correctly identified the last team to touch the ball before it exited the field, among the times when it correctly called an out

	QF	SF	F
true goals	3	2	2
false chip goals	1	3	1

Table 9 The counts of detected goals from the games. The only errors were false positives due to chip kicks

The autoref detected several occasions in the games where the maximum ball speed rule was violated, none of which resulted in corresponding calls by the human referee. In the quarterfinal game, our opponents violated the rule three times in the first half, kicking the ball in excess of 9 m/s, well above the stated limit; in the second half, we ourselves kicked a ball slightly too fast. There was also one spurious identification in each half, caused by false vision reports of a ball when a human stepped onto the field. (In practice, these would not be likely to cause any issues, since there will be humans only on the field while the game is stopped. Of course, it is also possible to improve the detector to filter out such detections to begin with.)

7.2.3 Goals scored

Finally, we tested the detection of goals scored. Chip kicks are a major confounding factor here as well; many kickoffs are taken by simply chip kicking the ball directly over the opponent goal, which looks rather like the ball going into the goal.

The detector correctly identified all of the goals that actually occurred in the games we examined, and there were no false positives apart from the ones caused by chip kicks.

8 Future work

The autoref in its current state can run a game from start to end while making the majority of the calls that should be made. However, there are still many important rules that need to be implemented for a reasonably complete game. Probably the most important of these are the rules regarding collisions between robots and touching the ball while inside the defense area. The rules that are concerned with the "intentions" of teams, while less essential, will probably present a bigger challenge.

At a lower level, the obstacles mentioned in Section 6.6 will need to be overcome for a truly satisfactory autoref; solutions to these problems would be helpful for both the autoref and our competitive team code, so we will continue to focus efforts in their direction. The chip kick detection (and, more generally, state estimation of the ball) especially would provide a benefit to performance. Even the event detectors that work could be put onto a more principled footing, e.g., by implicitly incorporating a further probabilistic model or explicitly reasoning about the belief state of the Kalman filter.

There are also human interface concerns that will also be important to address if an autoref is to become widely adopted; namely, that the autoref should be capable of both providing explanations for its calls and also gracefully allowing humans to override incorrect calls. As long as humans are still involved, it is important to ensure that they do not feel left out of the proceedings, and displaying explanations of its calls will make the autoref is a helpful tool rather than an inscrutable black box. The form of explanations that we envision is a sort of instant replay; the autoref could use an attached display to show animations of the parts of the preceding gameplay that led to the call in question.

Although we have striven to make sure that the autoref can handle situations correctly as often as possible, there may always be cases where it gets stuck or consistently handles something incorrectly, and in those cases, it will be necessary to have a human in the loop to bring it back on track. Even if the autoref is made as correct as it can be, the restricted input data available from SSL-Vision still make it impossible to detect, e.g., a robot which has broken and is leaving parts on the field. This is like the "Puppet Master" in CMCast (Veloso et al 2008), but it will probably be necessary to provide a more extensive interface, to account for the more detailed state that the autoref maintains.

We have performed extensive testing of our autoref within its domain, but we currently lack an objective comparison of our framework with others designed for similar tasks. In order to better understand the strengths and limitations of the approach described here, we would like to use other frameworks to implement the rules of the RoboCup SSL, and then compare the resulting referee systems in areas such as stability, adherence to the rules, and implementation clarity.

Finally, apart from the processing concerns, there is the issue of ball placement. We have used both team robots and a specialized robot to automatically place

the ball, but both methods have shortcomings: for the first case, the inability to solidly hold the ball in the first case, and, for the second case, given software limitations and the design of the current version of the robot, low speed and difficulty of control.

9 Conclusion

This paper describes a formalization of the RoboCup SSL, a complex, dynamic game, as a hybrid automaton based on its rules, as well as an algorithm for using observations from robots to estimate the evolution of such automata over time. The algorithm is based on a domain-specific set of event detectors which cooperate to advance the state of the automaton; although we studied and described the set of events appropriate to refereeing for the SSL, the main algorithm could apply to any such hybrid automaton.

To demonstrate the viability of the algorithm and event detectors, we implemented all of the necessary algorithms and applied them to the task of refereeing hundreds of simulated full SSL games. This allowed our team to perform new qualitative evaluations of soccer performance.

A Source code

Source code for our implementation of the system described here may be found at https://github.com/dzhu/ssl-autoref.

B Transition tables

Tables 10 to 18 include the full details of the transitions between the locations shown in Figure 6. Each table describes the set of transitions associated with one of the pairs of locations (i.e., one of the edges in the diagram). The first column of each table describes the conditions under which that transition occurs; the second indicates the command that is transmitted when it occurs; the third indicates the changes that are made to the variables of the hybrid automaton when the transition occurs. A dash in the first column indicates that the transition is always immediately taken when the automaton is in the originating location of the edge (this only occurs for the transition out of the INIT location).

References

Alur R, Courcoubetis C, Henzinger TA, Ho PH (1993) Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Hybrid Systems, Lecture Notes in Computer Science, Springer, pp 209–229

Arenas M, Ruiz-del Solar J, Norambuena S, Cubillos S (2009) A robot referee for robot soccer. In: RoboCup 2008: Robot Soccer World Cup XII, Springer, pp 426–438

Bruce J, Veloso M (2003) Fast and accurate vision-based pattern detection and identification. In: Proceedings of the 2003 IEEE International Conference on Robotics and Automation, IEEE, vol 1, pp 1277–1282

Bruce J, Balch T, Veloso M (2000) Fast and inexpensive color image segmentation for interactive robots. In: Proceedings of the 2000 IEEE/RSJ Conference on Intelligent Robots and Systems, IEEE/RSJ, pp 2061–2066

Chen M, Dorer K, Foroughi E, Heintz F, Huang Z, Kapetanakis S, Kostiadis K, Kummenje J, Murray J, Noda I, Obst O, Riley P, Steffens T, Wang Y, Yin X (2003) RoboCup Soccer Server Users Manual. http://sourceforge.net/projects/sserver/files/rcssmanual/9-20030211/

Egerstedt M (2000) Behavior based robotics using hybrid automata. In: Lynch N, Krogh B (eds) Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol 1790, Springer Berlin Heidelberg, pp 103–116

Fierro R, Das AK, Kumar V, Ostrowski JP (2001) Hybrid control of formations of robots. In: Proceedings of the IEEE International Conference on Robotics and Automation, 2001, IEEE, vol 1, pp 157–162

Henzinger TA (2000) The theory of hybrid automata. In: Inan M, Kurshan R (eds) Verification of Digital and Hybrid Systems, NATO ASI Series, vol 170, Springer Berlin Heidelberg, pp 265–292

Käppeler UP, Zweigle O, Häußermann K, Rajaie H, Tamke A, Koch A, Eckstein B, Aichele F, DiMarco D, Berthelot A, Walter T, Levi P (2010) RFC Stuttgart team description 2010. ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/INPROC-2010-97/INPROC-2010-97.pdf

Klavins E, Koditschek DE (2000) A formalism for the composition of concurrent robot behaviors. In: Proceedings of the IEEE International Conference on Robotics and Automation, 2000, IEEE, vol 4, pp 3395–3402

Mosterman PJ (1999) An overview of hybrid simulation phenomena and their support by simulation packages. In: Hybrid Systems: Computation and Control, Springer Berlin Heidelberg, pp 165–177

Niemüller T, Lakemeyer G, Ferrein A, Reuter S, Ewert D, Jeschke S, Pensky D, Karras U (2013) RoboCup Logistics League sponored by Festo: A competitive factory automation testbed. In: Proceedings of the 16th International Conference on Advanced Robotics — 1st Workshop on Developments in RoboCup Leagues

Schoenmakers F, Koudijs G, Martinez CL, Briegel M, van Wesel H, Groenen J, Hendriks O, Klooster O, Soetens R, van de Molengraft M (2013) Tech United Eindhoven team description 2013: Middle Size League. http://www.techunited.nl/media/files/TDP2013.pdf

Simon D (2006) Optimal state estimation: Kalman, H infinity, and nonlinear approaches. John Wiley & Sons

Srinivasa S, Berenson D, Cakmak M, Collet A, Dogar M, Dragan A, Knepper R, Niemueller T, Strabala K, Vande Weghe M, Ziegler J (2012) Herb 2.0: Lessons learned from developing a mobile manipulator for the home. Proceedings of the IEEE 100(8):2410–2428

SSL Technical Committee (2015) Laws of the RoboCup Small Size League 2015. http://wiki.robocup.org/wiki/File:Small_Size_League_-_Rules_2015.pdf

Tanaka K, Nakashima H, Noda I, Hasida K, Frank I, Matsubara H (1998) MIKE: An automatic commentary system for soccer. In: Proceedings of the International Conference on Multi Agent Systems, IEEE, pp 285–292

Vail D, Veloso M, Lafferty J (2007) Conditional random fields for activity recognition. In: Proceedings of the 6th Interna-

Condition	Command	Effect
—	STOP	call := KICKOFF is-kickoff := true stage := PRE-FIRST

Table 10 INIT to GAME_OFF transition. This transition simply sets up the initial values of the variables and sends an initial STOP command to make the robots get in position.

Condition	Command	Effect
ball enters goal of team T goal-allowed == true	GOAL(!T)	kick-team := T call := KICKOFF
ball gets stuck during play	STOP	call := START
robot R commits a minor foul	STOP	kick-team := !team(R) call := INDIRECT
robot R commits a serious foul	STOP	kick-team := !team(R) call := DIRECT
robot R commits a serious foul in its own defense area	STOP	kick-team := !team(R) call := PENALTY
kicker touches ball	STOP	kick-team := !team(kicker) call := INDIRECT
ball exits field	STOP	kick-team := !touch-team call := INDIRECT

Table 11 GAME_ON to GAME_OFF transitions. These are the transitions that cause active play to be temporarily stopped so a kick can be set up.

Condition	Command	Effect
robots settle call == START stage ∈ {PRE-FIRST,PRE-SECOND}	call(kick-team)	stage := next-stage(stage) stage-end := time + stage-length(stage) goal-allowed := true kick-deadline := time + 10
robots settle call == START stage ∉ {PRE-FIRST,PRE-SECOND}	call(kick-team)	goal-allowed := true kick-deadline := time + 10
robots settle call == DIRECT	call(kick-team)	goal-allowed := true kick-deadline := time + 10
robots settle call == INDIRECT	call(kick-team)	goal-allowed := false kick-deadline := time + 10

Table 12 GAME_OFF to SETUP transitions. These are the transitions that occur when the robots are ready to take a kick; the autoref's response varies somewhat depending on what kind of kick is to be taken.

Condition	Command	Effect
kick is taken by robot R with no infractions occurring	—	kicker := R touch-team := team(R) is-kickoff := false

Table 13 SETUP to GAME_ON transition. This transition represents the return to active gameplay after a free kick is correctly taken.

Condition	Command	Effect

robot `R` touches ball
`R` != `kicker`
	—	`kicker` := `null`
		`touch-team` := `team(R)`
		`goal-allowed` := `true`

Table 14 GAME_ON to GAME_ON transition. When a robot besides the taker of the last free kick touches the ball, this transition updates the variable representing which robot was the last to touch the ball and forgets the taker of the kick, since it is no longer relevant.

Condition	Command	Effect
robots settle `call` ∈ {`KICKOFF`, `PENALTY`}	`call(kick-team)`	`call` := `START`
min(`y-times(T)`) < 0	—	remove minimum element of `y-times(T)`

Table 15 GAME_OFF to GAME_OFF transitions. The first transition sends the next command to prepare for a kickoff or penalty kick. The second checks whether the oldest yellow card for a team has expired, and removes it from the list of cards if so.

Condition	Command	Effect
`time > stage-end` `stage` ∈ {`FIRST`,`SECOND`}	`stage-end-cmd(stage)`	`stage` := `next-stage(stage)` `stage-end` := `time + stage-length(stage)`
timeout requested by team `T` `timeouts-left(T) > 0`	`TIMEOUT`	`timeouts-left(T)` := `timeouts-left(T) - 1` `timeout-team` := `T`

Table 16 GAME_OFF to BREAK transitions. The first transition checks whether the time for a half has run out, and the next stage should therefore begin; the second occurs when a team signals for a timeout (a request which can only be granted while the game is stopped). In either case, the robots afterward are totally halted.

Condition	Command	Effect
`time > stage-end`	`stage-end-cmd(stage)`	`stage` := `next-stage(stage)` `stage-end` := `time + stage-length(stage)`
`timeout-time`(`timeout-team`) = 0	`STOP`	—
timeout end requested by `timeout-team`	`STOP`	—

Table 17 BREAK to GAME_OFF transitions. These transitions occur when the game resumes from being totally halted, either because the current stage has ended, the timeout team called an end to the timeout, or the timeout team ran out of time.

Condition	Command	Effect
`time > kick-deadline`	`STOP`	`call` := `START`
kick is taken kicker's teammate is near opponent defense area	`STOP`	`kick-team` := `!kick-team` `call` := `INDIRECT`
kick is taken opponent is near ball	`STOP`	`call` := `INDIRECT`
kick is taken `is-kickoff` == `true` some robot is on the wrong side of the field	`STOP`	—

Table 18 SETUP to GAME_OFF transitions. These correspond to the rules that restrict the conditions on the field when a kick is taken; if one of these conditions is violated when the kick is taken, the game stops rather than restarting.

tional Joint Conference on Autonomous Agents and Multiagent Systems, ACM, p 235

Veloso M, Armstrong-Crews N, Chernova S, Crawford E, McMillen C, Roth M, Vail D, Zickler S (2008) A team of humanoid game commenters. International Journal of Humanoid Robotics 5(03):457–480

Veloso M, Biswas J, Cooksey P, Klee S, Mendoza JP, Wang R, Zhu D (2015) CMDragons 2015 extended team description. http://robocup2015.oss-cn-shenzhen.aliyuncs.com/TeamDescriptionPapers/SmallSize/RoboCup_Symposium_2015_submission_81.pdf

Voelz D, André E, Herzog G, Rist T (1999) Rocco: A RoboCup soccer commentator system. In: RoboCup-98: Robot Soccer World Cup II, Springer, pp 50–60

Zhu D, Biswas J, Veloso M (2015) AutoRef: Towards real-robot soccer complete automated refereeing. In: RoboCup 2014: Robot World Cup XVIII, Springer, pp 419–430

Zickler S (2010) Physics-based robot motion planning in dynamic multi-body environments. PhD thesis, Carnegie Mellon University, Thesis Number: CMU-CS-10-115

Zickler S, Laue T, Birbach O, Wongphati M, Veloso M (2009) SSL-Vision: The shared vision system for the RoboCup Small Size League. In: RoboCup 2009: Robot Soccer World Cup XIII, Springer, pp 425–436