

# Action Selection via Learning Behavior Patterns in Multi-Robot Systems

Can Erdogan and Manuela Veloso

Carnegie Mellon University

Pittsburgh, PA 15213-3890

cerdogan@cmu.edu veloso@cs.cmu.edu

## Abstract

The RoboCup Small Size League robot soccer competitions have successfully taken place for thirteen years with autonomous systems where a combination of centralized perception and control, and distributed actuation takes place. In a given game, teams of five robots move at high speeds in a limited space, actuating a golf ball, aiming to score goals. Although the teams perform in a compelling way in principle, running pre-planned strategies, adapting in real-time to the adversarial teams is still a big challenge. In this paper, we introduce a representation that models the spatial and temporal data of a multi-robot system as instances of geometrical trajectory curves. We then explain how to model the behavior of a multi-robot system by implementing a variant of agglomerative hierarchical clustering. Next, we provide an algorithm that classifies a behavior concurrently as it occurs, with respect to a given clustering. Subsequently, we define an algorithm that autonomously generates counter tactics. We evaluate our work on logs from real games and in simulation.

## 1 Introduction

An interesting multi-robot system is the RoboCup Small Size league. In this domain, two teams of 5 robots engage in a soccer game, moving fast at 2m/s in a confined space of 6m by 4m. Each team is controlled by a central computer that receives visual data from two overhead cameras 60 frames per second. The visual data comprises of location and orientation information of each robot and the location of the ball.

Through our experience in the domain, we make the following conjecture: Each team has a finite set of strategies and, in a game, they choose to commit to a particular one depending on the position of the ball and other features of the game. For instance, such a strategy may include a robot preserving the possession of the ball while two others position themselves to receive a pass.

One of the features in OurTeam system is the extensive logging technique that is used to record the events in a game. In particular, the locations of the robots and the ball is stored at every frame, resulting in an extensive amount of data. Given

the game logs, one wonders if this data could be used to learn the underlying strategies that shape the behavior of a team.

To challenge this problem, we extract a set of trajectories from the game logs where a trajectory is defined as a sequence of timestamped points in multidimensional space. We select only the sections of the log where the ball possession is awarded to the opponent team and a passing motion is about to be executed. The problem is thus reduced to finding common patterns in a set of trajectory observations.

We first formalize the notion of similarity between sets of trajectories and then contribute a clustering technique to identify common patterns. The key idea we build upon is that trajectories can be represented as images and thus, shape matching algorithms can be utilized to compute distances. In our experiments, we observe that this framework can indeed identify the behavior patterns that underlie the strategies of 3 teams that OurTeam played against in RoboCup 2010.

The next question is whether we can predict how a behavior will shape in the future by matching it to a similar observation that has already been made. To this end, we contribute a heuristic that iteratively adjusts the durations of previous observations so that comparisons with the ongoing behavior are also feasible in time space. Our experiments show that we can identify behavior patterns with %70 success by watching less than %30 of any observation.

The final section of our work focuses on whether OurTeam can autonomously generate counter tactics and execute them successfully before the behavior pattern that it recognizes reaches an end. We completed experiments simulating the behavior of 3 opponent teams of OurTeam in RoboCup 2010. Our results suggest that if an opponent team does not change its commitment after detecting the counter tactic, the preemptive action succeeds %70 to %80 of the time.

To the best of our knowledge, our system is the first one to model the behavior of a multi-robot system and attempt to take actions to change its behavior. We note that although we base our work in the robot soccer domain, the algorithms we provide are general and applicable to other multi-robot domains. We organize the rest of the paper as follows. First we continue with the related work in the trajectory analysis domain. In Section II, we formalize the notion of similarity between behavior patterns of multi-robot teams and provide an algorithm to cluster similar patterns. Next, in Section III, we present a heuristic algorithm to classify behavior patterns on-

line and introduce a framework that autonomously generates counter tactics. Finally, in Section IV, we review the main contributions of our study and discuss future work.

We discuss related work in the subject of trajectory analysis that is closely related to two research areas: (i) similarity metrics between trajectories, and (ii) clustering techniques. Concerning similarity metrics, different domain-dependent approaches have been studied. Li et al. [2010] base their distance metric on a set of functions defined over the individual line segments to efficiently create compact clusters of trajectories, whereas, Hwang et al. [2005], make use of points of interests on their trajectories, such as sharp turns as they work on clustering trajectories on road networks. Other common approaches are based on the Longest Common Subsequence model [Vlachos et al., 2002] and Dynamic Time Warping [Corman et al., 1990]. These approaches, in addition to the spatial matching, also focus on the temporal matching of data points in the trajectories. The drawback of these algorithms is mostly their sensitivity to input parameters and potential to be computationally expensive. Shape matching approaches also exist, using geometric distances. In this paper, we implement the Hausdorff distance metric [Rote, 1991]. Similarly, Shao et al. [2010] propose a modification of the Hausdorff distance to match trajectories with different updating policies.

Regarding clustering techniques, most popular approaches are density-based [Li et al., 2010], hierarchical [Kumar et al., 2002], grid-based [Sacharidis et al., 2008] and based on neural networks [Hu et al., 2003]. Density-based and grid-based approaches are generally used for larger datasets, whereas, neural networks need to be trained. In this work, we adopted an agglomerative hierarchical technique where we provide an algorithm to create partitions from the hierarchical tree representation. Our algorithm is a modification of the approach pursued by Kumar et al. [2002], starting with each cluster as an element and building up to partitions by a series of merges.

A third categorization is the underlying purpose of the trajectory analysis. For instance, the studies of Li et al. [2010] and Shao et al. [2010] are motivated to model the motion of the objects they analyze whereas Hu et al. [2003] and Sacharidis [2008] attempt to predict future events such as traffic accidents by clustering online data. In our work, in addition to online prediction, we focus on action selection to take preemptive measures to manipulate future events.

In the next section, we provide a definition of behavior pattern that characterizes the behavior of a multi-robot system.

## 2 Definition of a Behavior Pattern

In the Small Size robot soccer domain, the visual data regarding the states of the robots and the ball in the field is obtained from two overhead cameras that transmit information 60 fps. Let  $\delta t$  be the frame period,  $1/60 = 0.016s$ . Additional data with respect to the state of the game, such as referee calls, timeouts and etc. are obtained from another computer named referee box. For a time frame  $t$ , the input data to the system can be summarized as: (i) location, orientation and team of each robot  $r \langle x_{r,t}, y_{r,t}, \theta_{r,t}, team_r \rangle$ , (ii) location of the ball  $\langle x_{b,t}, y_{b,t} \rangle$ , and (iii) the referee call  $r_t$ .  $r_t$  can be one of the following characters: {'F', 'I', 'P', 'K', 'S', 's'} which stand

for free kick, indirect kick, penalty, kickoff, start and stop commands. In OurTeam, in addition to processing the input data in real-time and transmitting commands to the robots, we also store the data in a log file for further inspection. Next, we discuss how we preprocess this data to extract useful sets of trajectories.

Given a time frame  $[t_0, t_n]$ , we define a **trajectory**  $T$  to be a vector of points recorded within that time frame, such that  $T(t_0, t_n) = \{(x_{i,t_0}, y_{i,t_0}) \dots (x_{i,t_n}, y_{i,t_n})\}$ . We represent the behavior of a robot within a time frame with the trajectory it follows throughout that duration. Next, we define an **episode**  $E = [t_0, t_n]$  as a time frame during which we analyze the behavior of an attacker team to deduce its offensive strategies. An episode starts when the ball possession is given to a team due to a free kick or an indirect kick, and then, the team has to make a pass due to game rules. Figure 1 depicts the behavior of a team within an episode.

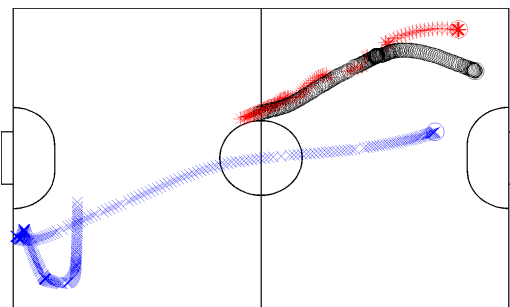


Figure 1: The behavior of a team throughout an episode where blue and black robots position for a pass and the ball (in red) is passed to the black one.

An episode *begins* at time  $t_0$  if the free or indirect kick command is given by the referee, following a duration of stop commands: for some constant  $c_1$ ,  $r_t = 's' \forall t \in \{(t_0 - c_1 * \delta t) \dots (t_0 - \delta t)\}$  and  $r_{t_0} \in \{'F', 'I'\}$ . In the time frame  $[(t_0 - c_1 * \delta t), (t_0 - \delta t)]$ , all robots must stay in 50cm away from the ball. With the referee signal  $r_{t_0}$ , an attacker robot  $R_A$  enters the 50cm radius zone to actuate the ball. Let  $BR$  be the radius of the ball and  $MR$  be the maximum radius of a robot. An episode is placed in the database unless:

1. the ball is not actuated by the kicker of a robot:  $\text{angle}((x_{b,t}, y_{b,t}) - (x_{A,t}, y_{A,t})) \neq \theta_{A,t}$  at time  $t$  such that  $\text{dist}((x_{b,t}, y_{b,t}), (x_{A,t}, y_{A,t})) \leq (BR+MR)$ ; or,
2. the game is stopped before the ball moves:  $r_l = 's'$  for a  $l \leq \text{argmin}_{t>t_0}(\text{dist}((x_{b,t_0}, y_{b,t_0}), (x_{b,t}, y_{b,t}))) > BR$ ; or,
3. robots stay stationary:  $\forall t \in \{t_0 \dots t_0 + c_2 * F\}$   $\max(\text{dist}((x_{i,t_0}, y_{i,t_0}), (x_{i,t}, y_{i,t}))) > MR$ .

We set the time related constants  $c_1$  and  $c_2$  to  $10 * \delta t \leq 0.2s$ .

Let  $t_m$  be the time the ball was actuated by the actuator robot  $R_A$  as computed in the second validity condition above. An episode *ends* at time  $t_n$  if and only if:

1. the ball collides with a robot after being actuated:  $\text{dist}((x_{i,t}, y_{i,t}), (x_{b,t}, y_{b,t})) \leq BR+MR$  for a  $t > t_m$ ; or,
2. the ball is out of bounds: for some  $t > t_m$ ,  $|x_{b,t}| > 3m$  or  $|y_{b,t}| > 2m$ .

Note that episodes can be extracted both from game logs and in real-time, for offline and online learning respectively.

The definition of an episode allows us to identify the sections of a game where we hypothesize that teams execute an instance of a behavior chosen from a predefined finite set. We define the behavior of a team in an episode  $E_k$  as a set of trajectories  $S^k = \{ \{ (x_{i,t_0^k}, y_{i,t_0^k}) \dots (x_{i,t_n^k}, y_{i,t_n^k}) \} \mid \text{robot } R_i \text{ is an active agent in } E_k \}$ . Without losing generalization, we assume that the observed team is attacking to the right side of the field. Then, robot  $R_i$  is an **active agent** in  $E_k$  if:

1. it manipulates the ball:  $\exists$  some  $t$  such that  $t_0^k < t < t_n^k$ ,  $\text{dist}((x_{i,t}, y_{i,t}), (x_{b,t}, y_{b,t})) < (\text{BR} + \text{MR})$ ; or,
2. it is at the opponent half of the field at some point in  $E_k$ :  $\exists$  some  $t$  such that  $t_0^k < t < t_n^k$ ,  $x_{i,t} > 0$ .

Figure 2 depicts an example of three active robots scoring a goal while two defensive robots do not take any action. Note that following Figure 3, we depict behavior of teams in episodes only with the active agents.

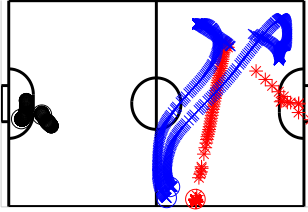


Figure 2: A set of trajectories from where two robots with blue trajectories position themselves to receive a pass. The red trajectory belongs to the ball. The robot that receives the pass, shoots at the ball. The robots with black trajectories do not have any impact in the scene.

We define a **behavior pattern** as a cluster of similar sets of trajectories  $S_i$ . Figure 3 demonstrate the behavior of robots in two episodes where the robots perform the same motion.

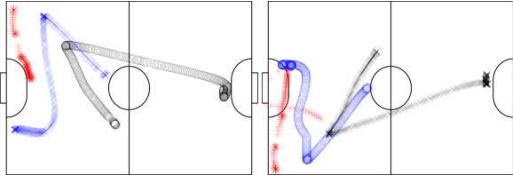


Figure 3: Two episodes of a team, during which the joint behavior of the robots are the same. Note that they are mirror images of each other horizontally.

### 3 Learning Behavior Patterns

In this section, we first formalize the notion of similarity between sets of trajectories; then present a clustering algorithm to extract common behavior patterns and finally, demonstrate experimental results.

#### 3.1 Similarity between Sets of Trajectories

Assume that we want to quantify the similarity between two sets of trajectories  $S_1$  and  $S_2$ , during two episodes  $E_1 = [t_0^1,$

$t_n^1]$  and  $E_2 = [t_0^2, t_n^2]$  respectively. For any set of trajectories denoted by  $S$ , let  $S[i]$  denote the  $i^{\text{th}}$  trajectory in  $S$  and let  $|S|$  be the number of trajectories in  $S$ . Note that we also refer to sets of trajectories in a given time frame  $[t_0, t_n]$  with the  $S(t_0, t_n)$  notation. First, we discuss the Hausdorff distance that computes the similarity of two trajectories.

Given two trajectories  $T_1$  and  $T_2$ ,  $H(T_1, T_2)$ , the Hausdorff distance between them is defined as:

$$H(T_1, T_2) = \max \left\{ \max_{p \in T_1} \min_{q \in T_2} E(p, q), \max_{p \in T_2} \min_{q \in T_1} E(p, q) \right\}$$

where  $E(p, q)$  is the Euclidean distance between points  $p$  and  $q$ . Note that the more similar two trajectories are, the smaller the Hausdorff distance is. The computation time is  $O(mn)$  where  $m$  and  $n$  are the number of points in  $T_1$  and  $T_2$  respectively. Next, we formulate the similarity function between two sets of trajectories.

Given the sets of trajectories  $S_1$  and  $S_2$ , each with  $n$  trajectories, one needs to first find the correspondence between the trajectories of the two sets that minimizes the Hausdorff similarity. Let  $\text{Perm}(n)$  be the set of all of the permutations of integers from 1 to  $n$ . A permutation  $P$  can be interpreted as a matching rule where the  $i^{\text{th}}$  trajectory in one set corresponds to  $P(i)^{\text{th}}$  trajectory in another. Then, the best matching  $M$  can be formulated as follows:  $M_{S_1, S_2} = \text{argmax}_{P \in \text{Perm}(n)} \sum_{i=0}^n H(S_1[i], S_2[P(i)])$ . So, a possible similarity function is:

$$\text{Sim}(S_1, S_2) = \sum_{i=0}^n H(S_1[i], S_2[M_{S_1, S_2}[i]]).$$

A given set of trajectories can be generated in 3 additional coordinates as mirror images of the robot locations about the horizontal and vertical axes of the field. To address this issue, we update the similarity function:

$$\text{Sim}(S_1, S_2) = \min_{F \in \text{Flips}} \sum_{i=0}^n H(F(S_1[i]), S_2[M_{S_1, S_2}[i]]),$$

where  $\text{Flips}$  is a set of functions where each function returns a symmetric match of the input input set of trajectories.

The last setback of this similarity function is the frequent calls made to the distance function directly proportional to the number of points in the trajectories of the two sets. As a solution, we reduce the length of the trajectories so that we consider every  $n^{\text{th}}$  data point with the function denoted as  $R$ :

$$\text{Sim}(S_1, S_2) = \min_{F \in \text{Flips}} \sum_{i=0}^n H(F(R(S_1[i])), R(S_2[M_{S_1, S_2}[i]])).$$

In summary, given two sets of trajectories, this formulation achieves the following properties: (i) finds the trajectories of the active agents, (ii) creates a matching between the trajectories of the sets, (iii) handles different inversion cases, and (iv) allows different sampling rates of the robot trajectories.

#### 3.2 Clustering Sets of Trajectories

Given a similarity function between the sets of trajectories, we partition them into clusters such that each cluster represents a behavior pattern. In Figure 4, we present the sets of trajectories we extracted from the episodes of a team in a tree obtained by hierarchical clustering. The lines in the labeled

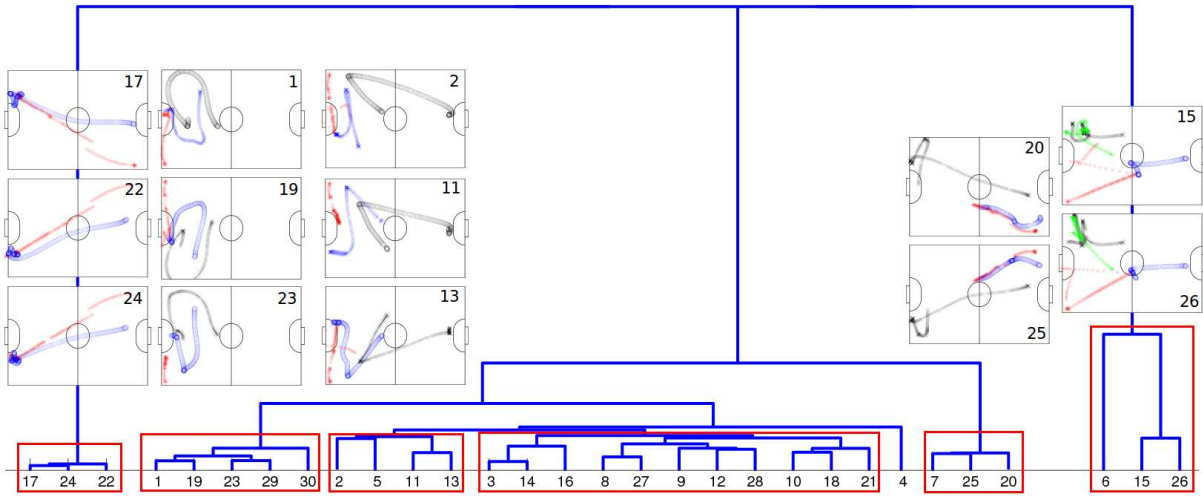


Figure 4: A sample clustering of sets of robot trajectories into behavior patterns. The red boxes represent the clusters obtained by hierarchical clustering. The trajectory sets in the same cluster are instances of the same behavior pattern.

boxes depict the trajectories of the robots and the ball (in red). The boxes around the leafs of the tree represent the clusters obtained and thus, the behavior patterns extracted from the sets of trajectories. For instance, trajectory sets 17, 22 and 24, clearly belong to the left-most cluster.

We implement a variant of hierarchical clustering due to two challenges imposed by our datasets. First, due to the small size of our data with 20 to 100 elements, outliers can significantly affect the results of partitional algorithms. Second, the variance in the densities of clusters we obtain from hierarchical algorithms show that such data can not be successfully clustered by density-based algorithms.

Our implementation takes two inputs, `minSize` and `maxSize`, that identify the minimum and maximum number of elements a cluster can have. Below we present the algorithm:

1. Place each element in the dataset into a cluster
2. Compute the distance between every cluster by taking the average of distances between each element.
3. Merge the closest pair of clusters if the number of elements of the new cluster is less than `maxSize`
4. Repeat steps 2-3 until number of clusters do not change
5. Remove any clusters that have less than `minSize`

We make two additions to the classic agglomerative hierarchical clustering technique. First, we bound the number of clusters by limiting the size of a cluster. Second, we postprocess the clustering to detect outliers, those with size smaller than `minSize`, and remove them.

### 3.3 Experimental Results

We provide experimental results that demonstrate that our framework can indeed find the behavior patterns of a team by observing its game play. We evaluate our work on real game logs, testing whether we can deduce patterns from both our opponents' and our own game play.

To quantify the quality of the clusterings obtained, we compare the results with the clusterings generated by ten human classifiers. The comparison of clusterings is based on the

Rand index, an objective criteria frequently used in clustering evaluation. For two clusterings  $C_i$  and  $C_j$  of  $n$  elements, with clusters  $\{c_{i1}, \dots, c_{in}\}$  and  $\{c_{j1}, \dots, c_{jm}\}$  respectively, we define two values  $p_{same}$  and  $p_{diff}$ .  $p_{same}$  is the number of elements in the same cluster and  $p_{diff}$  is the number of elements in different clusters in both  $C_i$  and  $C_j$ . The Rand index  $R(C_i, C_j)$  is then computed as  $(p_{same} + p_{diff}) / \binom{n}{2}$ . Note that if Rand index 1.0, then two clusterings are the same.

Table 1 below summarizes the clusterings obtained in our experiments by providing the average Rand Index computed between the output of our system and each of the ten human clusterings. Note that for TeamA, TeamB and TeamC, we set the `minSize` and `maxSize` values in the clustering algorithm to 5 and 15 respectively whereas for OurTeam, we increased the `maxSize` to 25 for the greater number of episodes.

Table 1: Clustering Success in RoboCup Games

Team	Episodes	Clusters	Ave. Rand Index
OurTeam	100	11	0.96
TeamA	30	8	0.87
TeamB	23	4	0.91
TeamC	14	2	0.94

## 4 Reacting to Behavior Patterns

We discuss online classification, the Non-Responsiveness assumption and action selection algorithm.

### 4.1 Online Classification

Let  $C$  be a clustering of sets of trajectories such that  $C = \{c_1, \dots, c_n\}$  and let each cluster  $c_i$  be annotated as multiple sets of trajectories:  $c_i = \{T_{i,1}, \dots, T_{i,m}\}$  where  $m$  is the size of  $c_i$ . Let  $t_n$  be the current time. Assume that the beginning of an episode was detected at time  $t_0 < t_n$  and the associated set of trajectories is being recorded. The episode will reach an end at time  $t_k > t_n$ . The objective is to place  $S(t_0, t_n)$  in clustering  $C$ , as if the entire set was observed in the time

frame  $[t_0, t_k]$ . Thus,  $S$  should either be classified into some cluster  $c_i$  or placed in a new cluster  $c_{n+1}$ .

To obtain the distance between a set of trajectories  $S$  and some cluster  $c_i$  in  $C$ , we compute the similarity function  $\text{Sim}(S, S_{i,j})$  for each  $S_j$  in  $c_i$  and then, computing the mean of these values. However, the concept of time should be included in the similarity computation of a partial set and a completed one. Let  $t_S = t_n - t_0$  be the current duration of the episode associated with  $S$ . We make the following claim: if two sets of trajectories are to be clustered together, their durations should be similar. Indeed, Table 2 demonstrates the average durations and the standard deviation of behavior patterns of TeamA game play, supporting our claim.

Table 2: Durations of Behavior Patterns of TeamA

Cluster	1	2	3	4	5	6	7	8
Time	5.5	7.1	4.2	4.7	3.5	4.3	4.5	5.0
Stdev	.42	.26	.12	.08	.16	.26	.11	.58

We use the following heuristic: If a partial set  $S$  spanning  $t_S$  and a complete set  $S_c$  spanning  $t_{S_c}$  are to be compared, and  $t_S < t_{S_c}$ , the number of points of each trajectory in  $S_c$  should be shortened by the ratio  $r_{S,S_c} = t_S / t_{S_c}$ . Thus, we reformulate the similarity function in Section 3.1 as:  $\text{Sim}(S, S_c) = \text{Sim}(S, \text{red}(S_c, r_{S,S_c}))$  where  $\text{red}$  is the reduction function that removes the last  $(1-r_{S,S_c})^{th}$  portion of the trajectories in  $S_c$ . Figure 5 demonstrates how the comparison of a partial set of trajectories  $S$  with two complete sets  $A$  and  $B$  would proceed in time. As  $t_S$  increases, greater ratio values are computed for each set  $A$  and  $B$ . Note that the observed robot in set  $A$  reaches its final position and waits in the time frame  $t_S = [1.93, 4.15]$ , whereas  $B$  changes considerably with the inclusion of a second robot.

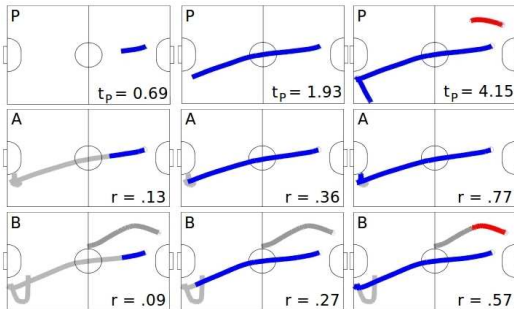


Figure 5: Incremental partial comparison of pattern  $P$  against behavior patterns  $A$  and  $B$ . At the end,  $t_p = 4.15$ s, the correct complete set  $B$  is identified.

Let  $t_S^c$  be the duration that the complete set  $S$  spans over and let  $t_S^n$  be the necessary amount of time trajectory set  $S$  must be observed to classify it correctly. Let  $o_P = t_S^n / t_S^c$  be the observed ratio of  $S$  that is necessary to classify  $S$ . Figure 6 presents the effect of different observation percentages on the correct classification of sets of trajectories for several teams. For instance, more than 70% of the trajectory sets of all teams are classified into correct behavior patterns by observing 30% of each trajectory set.

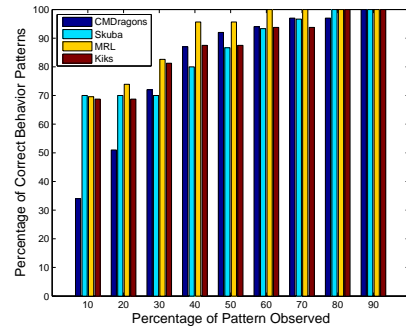


Figure 6: Graph depicting the effect of observation duration of partial patterns of a team on their correct classification.

## 4.2 Non-Responsiveness Assumption

After classifying behaviors concurrently, we want to take preemptive actions. To do so, we first introduce the concept of **Non-Responsiveness** as the inability of a system to adapt its behavior to external input once it has committed to the execution of that behavior. Figure 7 demonstrates two sets of trajectories that were observed within the same game, minutes apart. Each line represents the trajectory of a robot and a red circle depicts the beginning of a trajectory. We are interested in the behavior or robot  $R_1$  on the left. One would assume that it makes a circular motion because it tries to go to an open space, getting away from the red, defender robot  $R_3$ . However, this explanation does not hold true on the right, where  $R_1$  repeats the same motion although there are not any robots marking it. It could just go towards the goal directly. From these two cases, we claim that the blue team did not take the defense into account on the left scene, in the first place. We generalize the claim to other teams in RoboCup.

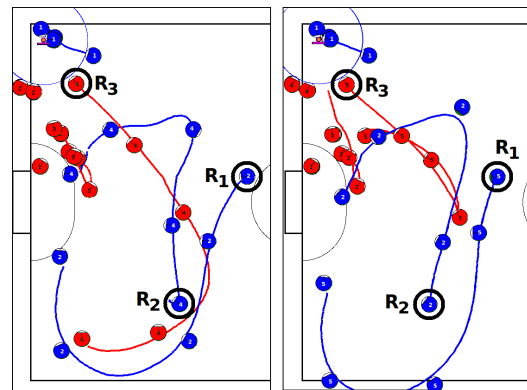


Figure 7: Two cases that help support the claim that the blue team is not responsive to the actions of the red defense.

## 4.3 Action Selection

Let  $C$  be a clustering as defined in Section 4.1. Assume for each cluster  $c_i = \{p_{i1}, \dots, p_{im}\} \in C$ , an optimal action  $A_{c_i}$  exists such that if a trajectory set  $T_{i,j} \in c_i$  is being observed and  $A_{c_i}$  can be executed, then a successful counter measure is taken. We define an optimal action as preventing the first pass between the actuator robot that initiates the game play and the opponent robot which receives the obligatory first pass

A policy is a mapping from the world states to valid actions in a domain. Let  $\Pi$  be the policy that governs the actions of an observer system. Let  $W_{t_n}$  be the state of the world at time  $t_n$  for which action  $A_{t_n}$  is chosen by policy  $\Pi$ . We propose that  $A_{t_n}$  should be replaced by the optimal action of some cluster  $c_i$ ,  $A_{c_i}$ , if and only if, a partial trajectory set  $S$  is observed at time  $t_n$  and classified into  $c_i$ . Simply put, if the observer system can predict the future states by matching a partial pattern with those in its database, then, it should overwrite its policy and attempt to take a specialized action for that case. We present the **SelectAction** algorithm to summarize this discussion. Note that we refer to functions such as *getObservedPattern* and *onlineClassifier* whose implementations were discussed in Section 2 with definitions of episodes and in Section 4.1 on online classification. Additionally, *Applicable* is a function that checks whether the preconditions of the action would hold until it is completed, i.e. if a robot has enough time to reach a position to intercept a pass.

**Algorithm SelectAction**

**Input:**  $\Pi$ : Policy;  $S$ : Current state;  $C$ : Clustering of behavior patterns, each cluster has an associated optimal action;

**Output:** Action to be taken in that state

1.  $P \leftarrow \text{state.getObservedPattern}();$
2. **if**  $P \neq \text{NULL}$
3.     **then**  $\text{cluster} \leftarrow \text{onlineClassifier}(C, P);$
4.          $\text{optAction} \leftarrow \text{cluster.getOptimalAction}();$
5.         **if**  $S.\text{Applicable}(\text{optAction})$
6.             **then return**  $\text{optAction};$
7. **return**  $\Pi(S);$

#### 4.4 Experimental Results

We ran experiments on real game data in the following manner. For each team, we first let our system process the previous games from the logfiles, learning the behavior patterns. Second, we simulate a new game where we run our system as usual, with the exception that visual data of opponents is from the log files. Only the detected episodes are replayed based on the **Non-Responsiveness** assumption. For each episode, we observe whether the new soccer program with the **SelectAction** algorithm intercepts the passes.

Note that in some episodes, the pass from the actuator might not reach a receiver robot due to an interception by the other team or to the inaccuracy of the actuator. Regardless, we ignore those cases since from the log files, we can not actually simulate the movement of the ball and create a collision by intercepting the ball. Table 3 summarizes the results.

Table 3: Intercepted Passes in Log Simulations

Team	Patterns	Interceptions	Success Rate (%)
TeamA	26	21	80.7
TeamB	13	10	76.9
TeamC	15	11	73.3

The results show that we can identify the strategy of an opponent team and successfully intercept the passes %70 to %80 of the time. A detailed analysis of the failed interception cases reveal that there are two sources of error: (i) inability of the size of the database to capture every case and (ii) the late classification of partial trajectory sets into the right clusters.

In the second case, even though the right optimal action is chosen, the robots do not have enough time to execute it.

## 5 Conclusion

In this paper, we formalized a similarity measure between behavior patterns executed by robot soccer teams. Additionally, we implemented a variant of the hierarchical clustering algorithms that can efficiently model observed behavior patterns. Moreover, we provided an algorithm that takes preemptive actions to manipulate the outcome of the observed behavior. We tested our work both on real game data and in simulation, and demonstrated the success of our methods. Regarding future work, we plan to weaken our assumption that teams commit strongly to the patterns they execute.

## References

- [Corman *et al.*, 1990] Thomas H. Corman, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.
- [Hu *et al.*, 2003] Weiming Hu, Xuejuan Xiao, Dan Xie, and Tienue Tan. Traffic accident prediction using vehicle tracking and trajectory. In *Intelligent Transportation Systems, 2003. Proceedings*. IEEE, 2003.
- [Hwang *et al.*, 2005] Jung-Rae Hwang, Hye-Young Kang, and Ki-Joune Li. Spatio-temporal similarity analysis between trajectories on road networks. In *ER Workshops*, pages 280–289. Springer-Verlag, 2005.
- [Kumar *et al.*, 2002] Mahest Kumar, Nitin R. Patel, and Jonathan Woo. Clustering seasonality patterns in the presence of errors. In *KDD '02 Proc. of the Eighth ACM SIGKDD Intr. Conf. on Knowledge Discovery and Data Mining*, pages 557–563. ACM, 2002.
- [Li *et al.*, 2010] Zhenhui Li, Jae-Gil Lee, Xiaolei Li, and Jiawei Han. Incremental clustering for trajectories. In *DAS-FAA 2010, Part II, LNCS 5982*, pages 32–46, 2010.
- [Rote, 1991] Gunter Rote. Computing the minimum hausdorff distance between two point sets on a line under translation. In *Information Processing Letters*. Elsevier North-Holland Inc, 1991.
- [Sacharidis *et al.*, 2008] Dimitris Sacharidis, Kostas Patroumpas, Manolis Terrovitis, Verena Kantere, Michalis Potamias, Kyriakos Mouratidis, and Timos Sellis. On-line discovery of hot motion paths. In *Proc. of the 11th Intr. Conf. on Extending Database Technology*, 2008.
- [Shao *et al.*, 2010] Fei Shao, Songmei Cai, and Junzhong GU. A modified hausdorff distance based algorithm for 2-dimensional spatial trajectory matching. In *Computer Science and Education (ICCSE), 2010 5th Intr. Conf. on*, pages 166–172. IEEE, 2010.
- [Vlachos *et al.*, 2002] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *Proc. of the 18th Intr. Conf. on Data Engineering*. IEEE, 2002.