Plan Execution Monitoring through Detection of Unmet Expectations about Action Outcomes

Juan Pablo Mendoza[1], Manuela Veloso[2] and Reid Simmons[3]

Abstract— **Modeling the effects of actions based on the state of the world enables robots to make intelligent decisions in different situations. However, it is often infeasible to have globally accurate models. Task performance is often hindered by discrepancies between models and the real world, since the true outcome of executing a plan may be significantly worse than the expected outcome used during planning. Furthermore, expectations about the world are often stochastic in robotics, making the discovery of model-world discrepancies non-trivial. We present an execution monitoring framework capable of finding statistically significant discrepancies, determining the situations in which they occur, and making simple corrections to the world model to improve performance. In our approach, plans are initially based on a model of the world that is only as faithful as computational and algorithmic limitations allow. Through experience, the monitor discovers previously unmodeled modes of the world, defined as regions of a feature space in which the experienced outcome of a plan deviates significantly from the predicted outcome. The monitor may then make suggestions to change the model to match the real world more accurately. We demonstrate this approach on the adversarial domain of robot soccer: we monitor pass interception performance of potentially unknown opponents to try to find unforeseen modes of behavior that affect their interception performance.**

I. Introduction

To make intelligent decisions, robots often use models of the effects of their actions on the world. Unfortunately, in sufficiently complex environments, it is infeasible to have the computational resources and perfect knowledge required to create completely accurate world models. This limitation may lead to divergence between planned actions and actual execution. It is thus necessary to monitor the execution of plans, and to correct the model as needed to enable robots to improve their performance.

We present an execution monitoring framework that enables robots to improve performance by detecting poorly modeled sets of situations and correcting their models accordingly. In particular, we address the problem of finding and adapting to *regions of a state-action feature space* in which action outcomes observed during execution deviate from the expectations used to select those actions. Furthermore, since robotics domains are intrinsically noisy, we are interested in *stochastic expectations* for which a single failed execution episode may not be indicative of a poor model.

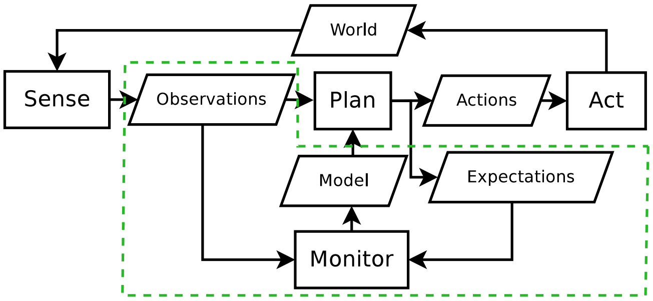


Fig. 1: High level framework for planning and execution. This paper focuses on the elements within the dashed line: We present a monitor that uses stochastic expectations generated by the planner, and observations from the world, to improve the model used in planning future execution.

Figure 1 illustrates the framework at a high level. This work focuses on the *Monitor* component of the framework, and its interaction, through inputs and outputs, with the *Plan* component. This feedback interaction can be summarized in the following looping steps:

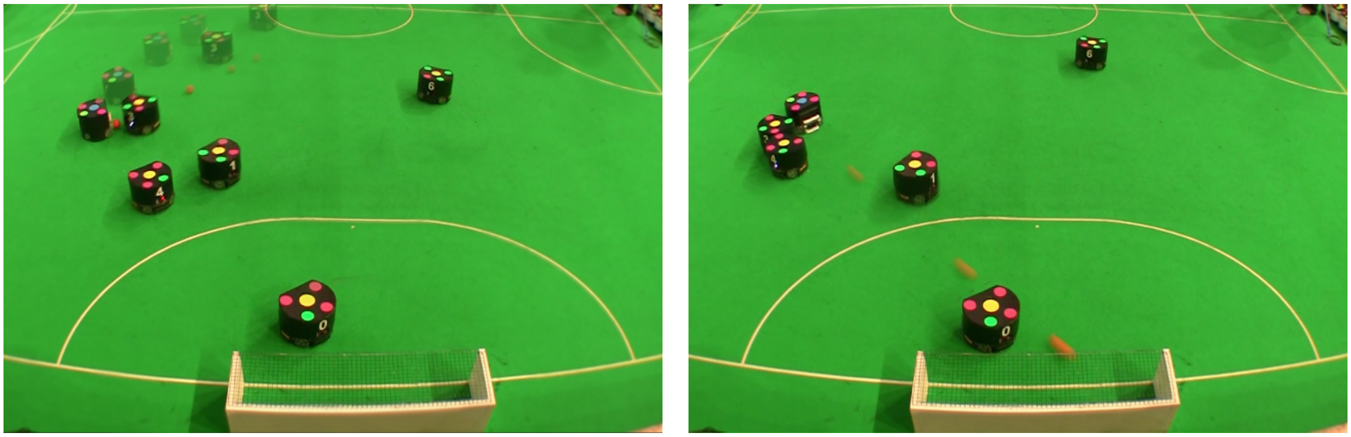
1) Create a model of the world that is generally accurate, but which may be suboptimal in some situations.
2) Create a plan to perform the desired task, based on the best available model. When a plan is created, in addition to generating a sequence of actions to perform, also generate a list of corresponding *expectations* about the results of those actions.
3) During execution, monitor whether the expectations were met in the real world by comparing them to *observations* obtained from sensing.
4) Find the conditions in which the real world is not well represented by the model. We represent such sets of conditions as *regions of a feature space* in which observations deviate from expectations in a statistically significant way.
5) Based on such findings, make corrections to the model in situations determined to be inadequately modeled.

Our application domain is a team of fast soccer-playing robots attempting to pass the ball to each other while preventing interceptions from opponent robots. Figure 2 illustrates how accurate ball interception modeling is crucial for success in the Small Size League of RoboCup [2], with robots capable of moving at over $3.5ms^{-1}$ and passing the ball at up to $8ms^{-1}$. However, in such adversarial domains, it is common to have incomplete information about the opponent's strategy and capabilities. We may be able to

[1] Juan Pablo Mendoza is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA jpmendoza@ri.cmu.edu

[2] Manuela Veloso is with the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA mmv@cs.cmu.edu

[3] Reid Simmons is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA reids@cs.cmu.edu



(a) Moving ball interception. Faded images show points along the trajectory of our robot, the ball, and a competing opponent robot.

(b) After a successful interception, our robot turns and scores with a narrow angle on goal.

Fig. 2: Goal scored by CMDragons during the quarter finals of RoboCup 2013. Accurate ball interception models of itself and of opponents allows our single robot (blue center dot) to gain control of a moving loose ball among several opponents (yellow center dots) to be able to score. (Thanks to J. Biswas for the interception and shooting algorithms [1].)

create a reasonably good model of how the world behaves (e.g., we may assume the opponent uses the same algorithm as our own robots to try to intercept passes). However, such models are likely to be poor predictors in some situations. In particular, opponents may have different strategies that they use in different circumstances; this means that there is likely to be a discrete boundary between the situations that are well-modeled and those that are not. Our goal is to find these boundaries, enabling our robots to improve their planning models and thus their passing performance.

II. Related Work

The problem of execution monitoring, well established in fields like industrial control, has gained increased interest in the robotics community [3], since robots need to be robust to failure in uncertain domains. An expectation-based monitor is one that monitors execution by comparing model-generated expectations of the world against corresponding observations received during execution. This type of monitor has been applied successfully to various robotics domains [4], [5]. While previous applications of expectation-based monitoring focus on detection and recovery from single failures, we focus on the detection of subtle, stochastic anomalies displayed over multiple trials, and model adaptation based on such detection.

Adaptation is essential for robots that act in changing or not fully modeled environments. Several Reinforcement Learning (RL) algorithms have addressed the problem of learning to perform well in a continuous environment that is not perfectly modeled. Model-free RL approaches, such as Q-Learning [6] and policy gradient descent [7], are capable of improving robot performance without explicitly modeling the world. While this generality is appealing and necessary in situations where modeling is impractical, learning tends to be less data-efficient and is not generalizable to different tasks within the same environment [8]. Model-based RL approaches learn, along with optimal policies, a transition model for the MDP that describes their world [9]. Our work is related to model-based RL in that we address the problem of learning a model to improve performance. However, in this paper, we focus on a framework that allows learning of different behavior modes, and we do not address the problem of exploration versus exploitation trade-off to achieve maximum reward. Furthermore, our framework requires and takes advantage of domain knowledge about expected, potentially long-term, effects of actions.

At the core of our algorithm is the detection of unmodeled modes of behavior through anomaly detection techniques, many of which have been applied in robotics and other fields [10]. In particular, we find statistical anomalies in collections of data. In this aspect, our work is related to time-series analysis [11], although we are more interested in data that is spatially related in some space of features of state-action space, rather than in purely temporal relationships.

III. Execution monitor design

The main contribution of this paper is the execution monitoring framework, and how it interacts with the planner. First, we describe how the planner generates the the necessary expectation information for plans to be monitored. We then describe how the monitor uses this information to find sets of situations in which the real world does not perform how the planner expected. Finally, we describe how the model might be updated to account for such detected discrepancies.

A. Planning with measurable expectations

We seek to improve the decision-making capabilities of a robot that operates in a state space S, and which can act upon its world through a set of actions A. We are interested in domains in which the planner makes decisions based on some quantifiable *expectations* of the world: the planner must be able to predict the results of applying action $a \in A$

in state $\boldsymbol{s} \in S$. We model these stochastic predictions as random observable variables $\boldsymbol{z} \in \mathbb{R}^n$ with parameters $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. For example, in the applications of this paper, we model the observables as normally distributed: $\boldsymbol{z} \sim \mathcal{N}(\tilde{\boldsymbol{z}}, \Sigma_{\boldsymbol{z}})$. Furthermore, the effects of action a need not be immediate, so the planner must also know when these observables can be evaluated. Formally, the planner and monitor have access to two functions:

$$\mathrm{Predict} : S \times A \rightarrow 2^{S \times \boldsymbol{\Theta}}$$
$$\mathrm{Observe} : S \times A \times S \rightarrow \mathbb{R}^n.$$

Function $\mathrm{Predict}$ models the expected (not necessarily immediate) effects of applying action a in state $\boldsymbol{s}$: $\mathrm{Predict}(\boldsymbol{s}, a)$ returns a prediction set $\mathcal{P} \in 2^{S \times \boldsymbol{\Theta}}$ of state-parameter pairs corresponding to the states in which the observable can be evaluated, and the parameters of the expected distribution of the observation at that state. For example, a soccer robot may predict that if it shoots the ball from its own goal (state $\boldsymbol{s}$) with speed v_0 (action a), then, when the ball exits the opposite end of the field (states in $\mathcal{P}$), it will have a speed normally distributed around $\tilde{v_1}$, dependent on the exit point (expectation $\boldsymbol{\theta} = (\tilde{v_1}, \sigma_v^2)$ in $\mathcal{P}$).

Function $\mathrm{Observe}$ is then used to verify or contradict such expectations during execution: if the robot finds itself in state $\boldsymbol{s_z}$ such that $(\boldsymbol{s_z}, \boldsymbol{\theta_z}) \in \mathcal{P}$, then $\boldsymbol{z} = \mathrm{Observe}(\boldsymbol{s}, a, \boldsymbol{s_z})$ is expected to be distributed according to $\boldsymbol{\theta_z}$. For the example above, when the ball reaches a position on the opposite end of the field, the robot can observe its true speed v_1 and evaluate its fit within the expected distribution $\mathcal{N}(\tilde{v_1}, \sigma_v^2)$.

Instead of a traditional planner in which only a sequence of actions (or a policy) is passed to the executing module, the planner can now also create a corresponding list of expectations of the outcomes such actions. Given that a plan (or policy) chooses to take action a when in state $\boldsymbol{s}$, an expectation e is defined as $e = (\boldsymbol{s}, a, \mathcal{P})$, such that $\mathrm{Predict}(\boldsymbol{s}, a) = \mathcal{P}$. These expectations are then used as input to the monitoring module in charge of verifying or contradicting their validity.

B. Monitoring expectations during execution

Once a plan has been created, the planner passes the list of expectations $e = (\boldsymbol{s}, a, \mathcal{P})$ to the monitor as execution begins. Algorithm 1 describes the procedure of the monitor during each step of execution.

The first step in monitoring consists of comparing the expected results created by the planner with the results $\boldsymbol{z}$ experienced during execution. To do this, for every expectation waiting to be verified, the monitor needs to check if the conditions for verification have been met – i.e., if the current state $\boldsymbol{s}_t$ is an element of the expectation termination states in $\mathcal{P}$. If so, then an actual observation is generated through the domain-specific function $\mathrm{Observe}$, which needs to look at the conditions in which the expectation was generated ($\boldsymbol{s}$, a), and the resulting state $\boldsymbol{s}_t$, to generate $\boldsymbol{z} = \mathrm{Observe}(\boldsymbol{s}, a, \boldsymbol{s}_t)$.

Once observations are generated, they are passed as input to an anomaly detector, which determines whether there are situations in which the predictive model does not correspond with the experienced reality. To do this, we use the Focused Anomalous Region Optimization (FARO) detector [12], described in detail in Section IV. The output of FARO is either an empty set, if no anomalies are present, or a set of states (represented as a region of feature space) in which expected behavior deviates significantly from experience, along with a maximum likelihood hypothesis of the true parameter value in that region.

Algorithm 1 Execution monitor procedure run every time step t of execution. Input: robot state $\boldsymbol{s}_t$, expectation list $\mathcal{E}$, set of regions $\mathcal{R}$ likely to be anomalous (initially empty).

```
function MONITOR($\boldsymbol{s}_t, \mathcal{E}, \mathcal{R}$)
                              ▷ First: add any new execution observations
    for each $e = (\boldsymbol{s}, a, \mathcal{P}) \in \mathcal{E}$ do
        if $\exists\,(\boldsymbol{s_z}, \boldsymbol{\theta_z}) \in \mathcal{P}$ s.t. $\boldsymbol{s_z} = \boldsymbol{s}_t$ then
            $\boldsymbol{z} \leftarrow \mathrm{Observe}(\boldsymbol{s}, a, \boldsymbol{s}_t)$
            add $(\mathrm{f}(\boldsymbol{s}, a), \boldsymbol{z}, \boldsymbol{\theta_z})$ to observations $\boldsymbol{Z}$.
            remove $e$ from $\mathcal{E}$
        end if
    end for
                                    ▷ Second: Find execution anomalies
    $(\mathcal{R}, \mathcal{A}) \leftarrow \mathrm{FARO}(\boldsymbol{Z}, \mathcal{R})$
                                        ▷ Third: Update planning model
    if $\mathcal{A} \neq \emptyset$ then
        $\mathrm{UpdateModel}(\mathcal{A})$
    end if
    return $(\mathcal{R}, \mathcal{A})$
end function
```

C. Modifying the planning model

Regions of anomaly detected by FARO are used to update the planning model accordingly. In our framework, this means modifying the $\mathrm{Predict}$ function into a new function $\mathrm{Predict}^+$ that incorporates information about anomalies found during execution. Thus, we use the output of FARO, which is a list of anomalies $\mathcal{A}$, each consisting of a region of feature space R_i of anomaly, as well as a maximum likelihood parameter deviation $\bar{\Delta}\boldsymbol{\theta}_i$ of observations within that region. Then, $\mathrm{Predict}^+$ is defined as

$$\mathrm{Predict}^+(\boldsymbol{s}, a) = \left\{(\boldsymbol{s_z}, \boldsymbol{\theta_z^+}) | (\boldsymbol{s_z}, \boldsymbol{\theta_z}) \in \mathrm{Predict}(\boldsymbol{s}, a)\right\}, \tag{1}$$

where

$$\boldsymbol{\theta_z^+} = \begin{cases} \boldsymbol{\theta_z} - \bar{\Delta}\boldsymbol{\theta}_i & \text{if } \exists R_i \in \mathcal{A} \text{ s.t. } \mathrm{f}(\boldsymbol{s}, a) \in R_i \\ \boldsymbol{\theta_z} & \text{otherwise} \end{cases} \tag{2}$$

That is, if $\mathrm{f}(s, a)$ is within a region of anomaly, predictions are shifted by the maximum likelihood shift determined by the anomaly detector, thus creating a new mode in the model.

IV. Detecting anomalous regions of state-action Feature space

The FARO anomaly detector [12] was designed to find regions of state space in which observations deviate significantly from expectations. Here, we use the FARO algorithm to detect anomalous regions of a state-action feature space, rather than only of state space directly. Algorithm 2 describes the FARO algorithm at a level of detail that fits the purposes of this paper. For a more detailed description of the algorithm, we refer the reader to [12].

Algorithm 2 FARO anomaly detector. Input: a list of observations $\boldsymbol{Z}$ and a set of regions $\mathcal{R}$ most likely to be anomalous. Returns: Updated set $\mathcal{R}$, and an anomalous region, if one is found.

```
 1: function FARO($\boldsymbol{Z} = (\mathbf{f}_i, \boldsymbol{z}_i, \boldsymbol{\theta}_i)_{i \in \{0,\ldots,t\}}, \mathcal{R}$)
 2:     $\mathcal{R} \leftarrow \mathcal{R} \cup \mathrm{r}(\mathbf{f}_t)$      ▷ Small region around latest obs.
 3:     $\mathcal{A} \leftarrow \emptyset$                      ▷ Initially, no detected anomaly
 4:     for $R \in \mathcal{R}$ do
 5:         Optimize $R$ into $R'$ such that
 6:             $\mathrm{anom}(R', \boldsymbol{Z}) \geq \mathrm{anom}(R, \boldsymbol{Z})$
 7:         $\mathcal{R} \leftarrow \mathcal{R} \cup R' \setminus R$
 8:         if $\mathrm{anom}(R', \boldsymbol{Z}) \geq a_{\max}$ then
 9:             $\mathcal{A} \leftarrow R'$                     ▷ Anomaly detected
10:         end if
11:     end for
12:     if $|\mathcal{R}| >$ capacity then
13:         $R^- \leftarrow \arg\min_{R \in \mathcal{R}} \mathrm{anom}(R, \boldsymbol{Z})$
14:         $\mathcal{R} \leftarrow \mathcal{R} \setminus \{R^-\}$
15:     end if
16:     return $(\mathcal{R}, \mathcal{A})$
17: end function
```

The FARO algorithm attempts to find regions of anomalous behavior using general optimization techniques. It conducts a parallel optimization on a few promising parametric regions of feature space (for this paper, ellipsoids), to find the one most likely to be a statistically significant anomaly.

The key computations of Algorithm 2 are the optimization of region R into R' (line 5), and the cost function $\mathrm{anom}(R, \boldsymbol{Z})$ used for it (line 6). As an optimization algorithm, FARO uses the cross-entropy method [13], although other optimization methods could be used instead. For the cost function to maximize, FARO uses the following:

$$\mathrm{anom}(R, \boldsymbol{Z}) = \frac{P(\boldsymbol{Z} | \text{behavior in } R \text{ is anomalous})}{P(\boldsymbol{Z} | \text{behavior in } R \text{ is nominal})}. \tag{3}$$

The region that maximizes this cost function is the one most likely to be anomalous. We can rewrite Equation 3 more specifically by assuming anomalies take the form of statistical deviations in the mean $\boldsymbol{\mu}$ of the expected distribution by some vector $\boldsymbol{\delta}$. In this case, assuming conditional independence among samples, we have:

$$\mathrm{anom}(R, \boldsymbol{Z} = (\mathbf{f}_i, \boldsymbol{z}_i, \boldsymbol{\theta}_i)) = \max_{\boldsymbol{\delta}} \frac{\prod_{\mathbf{f}_i \in R} P(\boldsymbol{z}_i | \boldsymbol{\mu}(\boldsymbol{\theta}_i) + \boldsymbol{\delta})}{\prod_{\mathbf{f}_i \in R} P(\boldsymbol{z}_i | \boldsymbol{\mu}(\boldsymbol{\theta}_i))}. \tag{4}$$

The value of the threshold value $a_{\max}$ (line 8) used to detect anomalies is obtained through Monte Carlo sampling [14] to achieve the desired tradeoff between false positive and false negative detections.

V. Monitoring pass interception in robot soccer

In this section, we describe the application of the framework presented in Section III to the robot soccer pass interception domain. The task consists of a team of soccer robots passing a ball to each other while preventing interceptions from opponent robots. The domain is inspired by the Small-Size League of Robot Soccer [2], where two teams of 6 robots each compete in a highly dynamic game of soccer. In this paper, we focus on the kicking robot's decision making, assuming a distributed architecture in which it has no influence over what actions its teammates take.

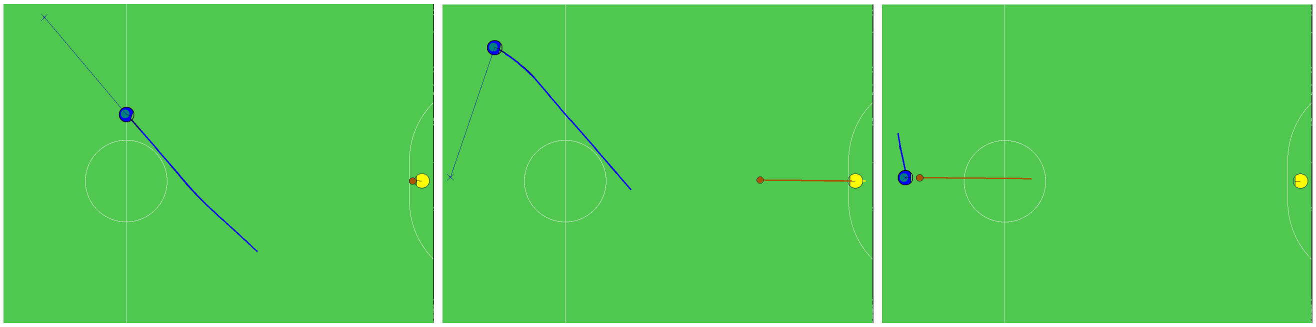
The full state space S of a robot soccer game involves over 80 continuous physical dimensions, to which one must add each team's internal state. The action space A for our problem is the 2-dimensional space of velocities at which a robot can pass the ball. For purposes of anomaly detection and correction, we capture the important features of the world in an 8-dimensional feature vector $\mathrm{f}(\boldsymbol{s}, a)$ per opponent robot: the ball position, its velocity in polar coordinates, and the opponent robot position and velocity in polar coordinates, both measured relative to the ball and the planned pass direction. While these features were enough for our demonstrative purposes, one may imagine using other features about the intercepting robot's state, or even about the rest of the robots on the field.

Since the focus of this paper is the monitoring of execution, rather than the planning, we apply a simple planning algorithm to the scenario: every time the robots need to pass the ball, they simply take the pass that maximizes the probability of one of our robots intercepting the ball before the opponents. Furthermore, we discretize the space of actions and search through all of them to pick the one that maximizes the expected reward.

With this planning scheme, our robots decide based on a model of the probability that a pass is successfully received by a teammate: $P(\text{success} \,|\, \boldsymbol{s}, a)$. To model this probability, we compute the predicted time $\tilde{\tau}$ that each robot in the field will take to intercept the ball. For this computation, we use the interception model of the CMDragons team [1]. We note that, while our own robots can actually use this interception model during execution, such that execution matches planning, we do not know what model the opponents use; this makes our model likely to be inaccurate in situations in which opponents behave differently from us. To map these interception times to a probability value, we compare the shortest predicted interception time $\tilde{\tau}_{\text{us}}$ among our robots, to the shortest predicted interception time $\tilde{\tau}_{\text{them}}$ among their robots:

$$P(\text{success} \,|\, \boldsymbol{s}, a) = \Phi\left(\frac{\tilde{\tau}_{\text{them}} - \tilde{\tau}_{\text{us}}}{\sigma}\right), \tag{5}$$

where Φ is the cumulative distribution of the standard normal distribution, and σ defines the uncertainty level in our predictions. Therefore, the probability of a successful pass smoothly changes from 0 when $\tilde{\tau}_{\text{us}} >> \tilde{\tau}_{\text{them}}$, to 0.5 when $\tilde{\tau}_{\text{us}} = \tilde{\tau}_{\text{them}}$, to 1.0 when $\tilde{\tau}_{\text{us}} << \tilde{\tau}_{\text{them}}$.



(a) Robot Y prepares to shoot, while B navigates to various locations.

(b) Robot Y shoots the ball, and B computes the optimal interception location.

(c) Robot B navigates to the chosen location to intercept the ball.

Fig. 3: Setup for ball interception tests. Yellow and blue circles depict robots from opposing teams (Y and B), while the orange circle depicts the ball. Thick lines indicate ball and robot trails, while the blue X indicates B's chosen target.

Having defined the problem and the planner, we now define the expectations $\mathcal{E}$ that will be monitored during execution. Our planner depends entirely on the model of interception times for each team, and our model of the opponent is usually the one that cannot be known in advance; because of this, we use the opponent interception time $\tilde{\tau}_{\text{them}}$ as the quantity to monitor. Every time the planner generates a passing action, it passes an expectation $e = (\boldsymbol{s}, a, \mathcal{P})$ to the monitor. Here, $\boldsymbol{s}$ and a are simply the state of the world and the chosen pass. Termination states in $\mathcal{P}$ are states in which a pass has just ended, determined by simple collision checks between the ball and the robots. Finally, the expected distribution of times is a normal distribution $\mathcal{N}(\tilde{\tau}_{\text{them}}, \sigma^2_{\tilde{\tau}})$. In this work, we allow $\sigma^2_{\tilde{\tau}}$ to be constant; however, this quantity could be learned and monitored as well.

During execution, the one-dimensional measured execution vector $\boldsymbol{z} = [\tau_{\text{them}}]$ would ideally represent the actual time the opponent robot took to intercept the ball. However, the ball may also be intercepted by one of our robots, or go out of bounds before any robot intercepts it. In these cases, if the pass finished at some time τ before the predicted interception time $\tilde{\tau}_{\text{them}}$ ($\tau < \tilde{\tau}_{\text{them}}$) passed, no observation is added to the monitor, as no information is gained about the accuracy of $\tilde{\tau}_{\text{them}}$. On the other hand, if the pass finished after time $\tilde{\tau}_{\text{them}}$ had passed ($\tau > \tilde{\tau}_{\text{them}}$), an observation is added with $\tau_{\text{them}} = \tau$; this is an underestimate of how long the opponent would have taken to intercept the ball, which correctly observes that $\tau_{\text{them}} > \tilde{\tau}_{\text{them}}$.

VI. Illustrative Result

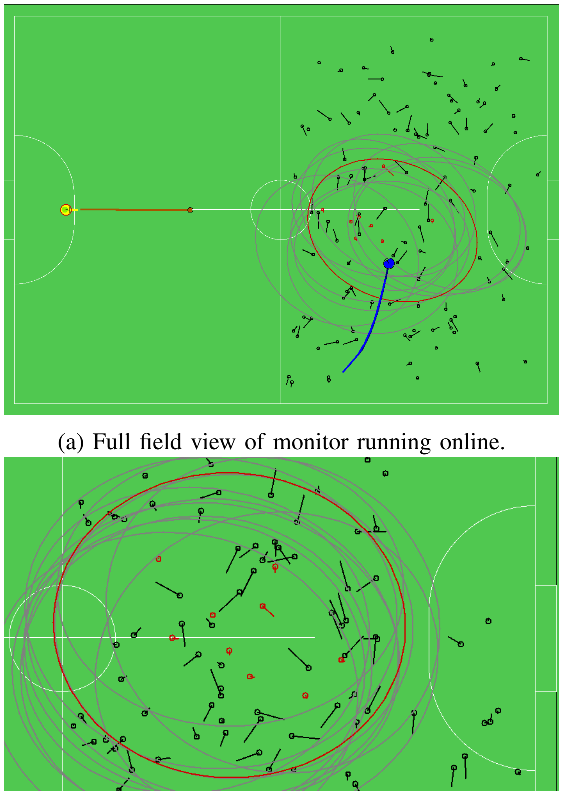
We deployed the execution monitoring framework, as described in Section V, and tested it with the setup illustrated in Figure 3: The yellow robot Y, from our team, has no teammates on its field, but it must perform passes. It was only allowed to pass from one starting location and in one direction, for ease of visualization below. Furthermore, since there is no chance of a successful pass, as there are no teammates on the field, all its available actions (pass speeds between $3ms^{-1}$ and $6ms^{-1}$) have the same expected reward, and so it chooses randomly among them. The opponent blue robot B continually navigates to various locations on its half of the field, but attempts to intercept any moving balls. This setup allowed us to obtain random samples of robot B's interception times starting with varying locations and velocities relative to the ball, and different ball speeds.

We ran this test multiple times on a PhysX-based simulation of our team, which includes robot models at the component level. Robot Y employed the FARO monitor, while robot B, whose model need not be known to Y, employed our regular ball-interception algorithms. The purpose of this experiment was to find out if the monitoring framework would find any anomalies in our own architecture; that is, find out if there were any unforeseen discrepancies between planning and execution.

After running the experiment multiple times, the monitoring framework repeatedly found an anomalous region of approximately the same shape. This shape, whose 2D projection is shown in Figure 4, contained states for which, at the moment the ball was passed, the opponent robot was already close to the trajectory of the ball along the perpendicular direction, and was either moving toward the ball's trajectory or had a small velocity component moving away. The average deviation $\tau_{\text{them}} - \tilde{\tau}_{\text{them}}$ between measured and expected interception time in this region was 0.3 seconds: the opponent robot was intercepting the ball significantly farther along the trajectory of the ball than predicted.

After analyzing the internal state of the intercepting robot, we realized that there was indeed a discrepancy between the algorithm used for planning and the algorithm used during execution. While the execution algorithm used the same computation to determine the fastest interception point, it contained another mode that was unaccounted for: If the robot was already on the path of a moving ball, then it ignored the computation of closest intercept point, and proceeded to stand its ground until the ball arrived to it. This mode was created to prevent oscillatory behavior and encourage a more stable reception of the ball, yet it was neglected by the planner.

This illustrative example reveals the value of our anomalous region-based monitor: The monitor was able to discover an unmodeled mode of the opponent's behavior. Even though



(a) Full field view of monitor running online.

(b) Close-up of detected anomaly in a different instance.

Fig. 4: Anomalous region detection result. Small circles with attached lines show observations of opponent robot location and velocity (in units of displacement over $0.1s$) when a pass starts. The red ellipse shows a 2D projection of the detected 8D anomalous region onto the space of opponent initial locations; data points that lie inside of the detected 8D ellipse are shown in red. Grey ellipses show other samples considered by FARO at the most recent execution step.

the robot does not understand the reasons behind the opponent's actions, it can understand and exploit the effects of this mode that are relevant to planning. Here, our robot can make a simple modification to its time estimate, as described in Section III-C, to improve the accuracy of the model. Our robots can thus benefit from this discovery by exploiting the region of sub-optimal performance (with respect to time) to make passes that would have seemed likely to fail before the model was corrected.

VII. Conclusion

This paper presents an execution monitoring framework for robots that make decisions based on measurable, stochastic expectations of how the world works. In particular, our framework is concerned with continuous multidimensional domains with potentially unknown modes of behavior, in which expectations of action outcomes are not realized during execution. The monitor finds these unknown modes by searching for regions of a state-action feature space in which execution deviates statistically significantly from expected action outcomes. Additionally, if an unmodeled mode is detected, the monitor also makes a simple suggestion on how to change the model to more accurately predict action outcomes in such mode.

The problem of detection and adaptation to unmodeled behavior modes is of particular interest in adversarial environments, since precise models of the opponents are rarely available. In robot soccer, not only do we not have exact models of the opponents, but opponents often intentionally reveal new strategies and techniques only at execution time. Monitoring our own robots as if they were opponents reveals a mode of behavior that was unaccounted for during planning. Finding such an unforeseen anomaly shows promise for the application of this monitoring framework in logs of games played during RoboCup, and perhaps in real-time during competition games.

In recent work [15], we show that an extension of the framework presented in this paper can detect multiple unmodeled behaviors, and correct the models accordingly. Empirical demonstrations have shown that such a framework can significantly improve performance in the complex robot soccer sub-task of keeping the ball away from opponent robots by passing it among teammates.

References

[1] J. Biswas, J. P. Mendoza, D. Zhu, B. Choi, S. Klee, and M. Veloso, "Opponent-driven planning and execution for pass, attack, and defense in a multi-robot soccer team," in *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, January 2014.

[2] Small Size Robot League. [Online]. Available: http://robocupssl.cpe.ku.ac.th/

[3] O. Pettersson, "Execution monitoring in robotics: A survey," *Robotics and Autonomous Systems*, vol. 53, no. 2, pp. 73–88, Nov. 2005.

[4] R. J. Doyle, D. Atkinson, and R. Doshi, "Generating perception requests and expectations to verify the execution of plans." in *AAAI*, T. Kehler, Ed. Morgan Kaufmann, 1986, pp. 81–88.

[5] G. D. Giacomo, R. Reiter, and M. Soutchanski, "Execution monitoring of high-level robot programs." in *Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann, 1998, pp. 453–465.

[6] C. Gaskett, D. Wettergreen, and A. Zelinsky, "Q-learning in continuous state and action spaces," in *Australian Joint Conference on Artificial Intelligence*. Springer-Verlag, 1999, pp. 417–428.

[7] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation." in *NIPS*, vol. 99, 1999, pp. 1057–1063.

[8] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *In International Conference on Robotics and Automation*, 1997.

[9] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, pp. 1–72, September 2009.

[11] E. Keogh and J. Lin, "Hot sax: Efficiently finding the most unusual time series subsequence," in *ICDM*, 2005, pp. 226–233.

[12] J. P. Mendoza, M. Veloso, and R. Simmons, "Focused optimization for online detection of anomalous regions," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, June 2014.

[13] R. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and computing in applied probability*, vol. 1, no. 2, pp. 127–190, 1999.

[14] M. Kulldorff, "A spatial scan statistic," *Communications in Statistics-Theory and methods*, 1997.

[15] J. P. Mendoza, M. Veloso, and R. Simmons, "Detecting and correcting model anomalies in subspaces of robot planning domains," in *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) (to appear)*, Istambul, Turkey, May 2015.