Intra-Robot Replanning to Enable Team Plan Conditions*

Philip Cooksey[1] and Manuela Veloso[2]

Abstract— **Individual team members are the building blocks of successful multi-robot teams in dynamic competitive domains. The current approach to designing a team is to divide the planning into a hierarchy by separating team coordination and task assignment – global planning – from task planning and execution – local planning. The global planner must make assumptions based on simplified models of dynamics and/or opponents, and as such certain conditions are assumed true when globally planning but are not always true at local execution time. In this paper, we describe several algorithms for intra-robot replanning that allow the individual robots to enable the conditions of their tasks. We then demonstrate improvements in task completion when the robots are capable of replanning their task(s) and their teammates' task(s) in a simplified robot soccer domain. We further show preliminary results on learning when to replan.**

I. Introduction and Related Work

Cooperative multi-robot team planners in competitive robot domains often have limited computational time, have poorly modeled dynamic objects, and have incomplete knowledge of their environment. These issues remain with the individual robots, however they can gather state information and learn to improve their execution by choosing the best fit replanning algorithms. Involving the team planner to incorporate more information on every robot is not effective or practical for highly dynamic domains with potentially many robots. Likewise, team planners often reduce information and complexity in order to make team planning feasible within dynamic domains, so additional information can hinder performance.

The standard approach to team planning in the literature is to use a hierarchy, thereby dividing up the planning problem involved in controlling a team of robots [1], [2], [3]. We follow the Skills, Tactics, and Plays (STP) hierarchy as described by [4]. This division of computation is used to simplify the problem of controlling a team in dynamic, competitive environments and allows planning to happen at different abstraction layers. Skills are low level repeatable algorithms that are specific to the domain. Tactics combine skills towards accomplishing a more complex task using finite-state machines, consider *passing the ball* which includes Skills like driving into the ball, positioning the ball, and kicking the ball. Plays guide the team towards

*This work is supported in part by AFRL and DARPA under agreement #FA8750-16-2-0042, ONR grant N00014-09-1-1031, and AFRL grant number FA87501220291. The views and conclusions contained in this document are those solely of the authors.

[1]Philip Cooksey is with Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. pcooksey@andrew.cmu.edu

[2]Manuela Veloso is with the School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. veloso@cs.cmu.edu

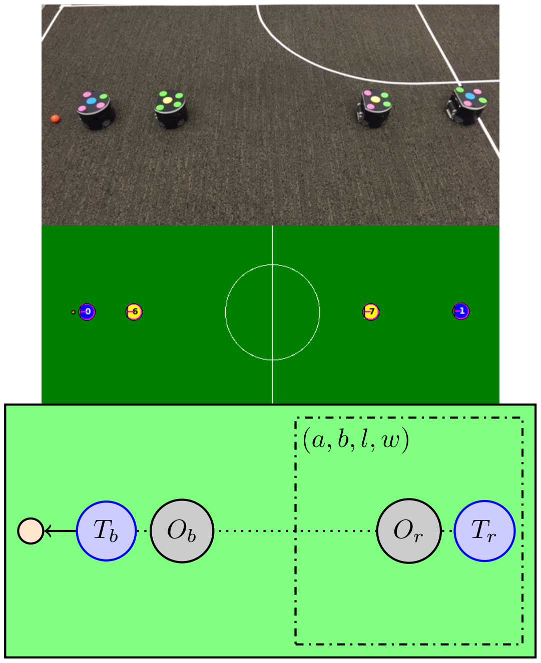

Fig. 1: Passing with marking: T_b needs to pass the ball to T_r within the zone (a,b,l,w). T_b is facing the ball with the direction arrow. Opponents O_b and O_r try to remain on the line between the other robots to block or intercept the pass. The top image is the physical robots, middle is the simulated robots, and bottom is a simplified representation that we will use in this paper.

their goal(s) and they define robot roles, positions, and a series of Tactics. Plays are changed based on defined state variables, i.e., number of robots and ball on defensive side. STP essentially separates planning into two levels, global (Plays) and local (Tactics).

In competitive dynamic domains, the global planner will assign a Play using incomplete information and simplified models of the environment. This missing information is often due to opponents' adversarial behavior but can also be attributed to the simplified models of the team members' abilities. A Play will only change when new information is presented at the global level that triggers a failure in its defined state variables, so failures at the local level might not be considered. This is due to the different requirements of the global planner and the local planning robots with respect to their abstraction in planning, i.e., to local information gains and/or the global planner not fully modeling the domain. We can especially see this in competitive domains like the

Small-Size Robot Soccer League (SSL).

The SSL league matches two teams of six omnidirectional robots and each team receives the same information (positions and headings) through overhead cameras [5]. Any further information must be generated by each team including velocities, predicting opponents, and the physics of ball movement at the expense of each team's computation. In STP, a Play may assign a pass between two robots for reasons unknown to the local robot. Still, the computation for completing that pass is left to the Tactic, and opponents may make that exact pass impossible to the assigned robot. Therefore, knowing that the global planner has made assumptions, we argue that robots can collect local information to learn when to use *intra-robot replanning* algorithms to enable conditions during execution to make a more successful team.

We note that Plays provide a fully instantiated *team plan* for each team member robot T_i. The robots execute their Tactics according to their assigned variables. The global planner may at any time change the Play due to new information, but there is no guarantee for changes in Plays due to faults observed only at the local level. The Play was created with the Tactics' conditions considered true and that the robot could complete its role using those Tactics. An implicit assumption was that a failure to complete a Tactic would lead to a global failure, which the global planner would then solve. If however a local failure does not cause an immediate global failure, then the robot will continue failing until complete global failure. In robot soccer, a local failure is a robot maintaining control of the ball but never finding an open pass, which results in it driving around in circles looking for an open pass. A global failure would be a missed pass or stolen ball.

In this paper, we investigate *intra-robot replanning* algorithms to enable conditions that are preventing locally succeeding at the assigned task. The topic of *intra-robot replanning* has recently been investigated and formalized in [6], which defines a more general model for replanning using different constraints. Replanning for individual robots has mostly been seen as a task of minimizing the changes to the current plan, and this same idea has been applied to multi-robot systems. However, in competitive domains, the robots have to compensate for the dynamic nature of the environment, so replanning must be quick and should be focused on enabling failed conditions with the highest chance of success. Optimality is often ill-defined in such complex domains so we focus on the robot accomplishing the assigned Tactic rather than failing, subsequently failing the team's objective. We contribute *intra-robot replanning* algorithms to enable the soccer robot to replan for failed conditions. We further demonstrate that the robot can learn which algorithm has the highest chance of success.

We highlight a few competitive domains that require *intra-robot replanning* but acknowledge that it can be useful in other domains like fleets of autonomous cars or drones coordinating towards a common goal while compensating for dynamic changes. In capture the flag [7], the team can have limited information about the opponent's team and their flag. The team plan that is created is limited to their available information. Assuming they can only globally plan on their side, they would need to locally replan for local changes in their information as well as their teammates'. Similarly, in robot soccer, the global planner has partial information about the opponent's behavior and it makes assumptions when creating Plays. The robot tasked with passing may need to replan its own position or the other robot's passing location to help it accomplish the pass. We will focus our efforts on a derivative of robot soccer to demonstrate the need for *intra-robot replanning* to enable conditions.

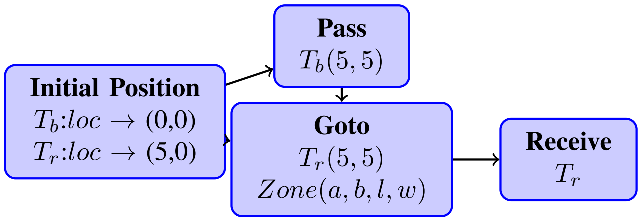

Fig. 2: A team plan for T_b to pass the ball to T_r.

II. Problem Domain

Passing with marking, shown in Figure 1, is a sub-domain of robot soccer where the task is for the robot with the ball, T_b, to pass the ball to the receiving robot, T_r, while opponents try to steal and/or intercept the ball. One opponent is always placed near T_b to block the initial pass. In our example, the opponents, O_i, are placed on the line between the two T_i to block and intercept the straight pass.

An example Play generated by the global planner is given in Figure 2. The Play is made of Tactics: *Pass*, *Goto*, and *Receive*. Each Tactic has instantiated variables defining the assigned robot and the variable(s) for that Tactic. For *Pass*, T_b is assigned to kick the ball to $(5, 5)$. For *Goto*, T_r is going to location $(5, 5)$ to *Receive* the ball and must stay within the $Zone(a, b, l, w)$ defining a box with the left top corner at location (a, b) with length l and width w.

The Play assigned the Role kicker to T_b and receiver to T_r. In this case, T_r is assigned the best passing location within its zone (the dashed-dotted square in Figure 1), but clearly O_r will attempt to intercept the ball. The assignment for the best passing location and the generation of the probability of success for that pass follows the approach in [8] and is outside the scope of this paper. The Play will often fail quickly from the opponents' interference. It can also fail due to randomness in the robot's performance and kicking ability in the physics simulator. There have been attempts to solve this issue in robot soccer as passing is a major requirement and machine learning has been a major focus of this research [9]. However, our approach is to design replanning algorithms that enable failed conditions and learn when to use each algorithm.

III. Individuals Enabling Conditions

In [10], they describe an approach called pass-ahead that tries to sync the arrival of the receiver with the ball so that they arrive at the passing location at the same time. This approach alone does poorly in the *passing with marking* domain because of the opponent O_b that blocks or steals the ball. In [11], they demonstrated the requirement of opponent aware algorithms for T_b to maintain possession of the ball, which improved pass-ahead's performance in the *passing with marking* domain (**Dribbling-Move** in Results Section). However, that algorithm does poorly in our example domain where an opponent was added to mark the receiving robot. The issue is that Dribbling-Move only focuses on replanning to enable one condition of the Tactic *Pass*, which is ball possession ($HasBall$). $HasBall$ is defined as true if the ball is closer to it than any other robot. And, the condition being failed is $Open$, which is defined true if the pass can be successful. $Open$ is ill-defined without knowledge of the opponents' intercept abilities and therefore is estimated by the global planner. Another issue is in T_b's ability to enable the $Open$ condition.

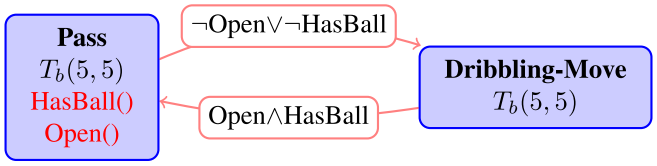

Fig. 3: *Intra-robot replanning* to enable Pass's condition $HasBall$ through dribbling. HasBall() and Open() are conditional functions (T/F) and switch what Tactic is used through the links.

Dribbling-Move enables the condition $HasBall$ by dribbling – the robot has a rotating bar in its forward direction, the arrow in Figure 1, that applies a back spin on the ball for maintaining possession – and can implicitly enable the condition $Open$ by moving the ball towards the passing location while circumventing nearby opponents. This does not solve the issue of opponents marking the receiver or the ill-defined $Open$ condition. For now, $Open$ can be defined as true if the probability of a successful pass, given by the team planner, is above the defined threshold γ. The partial Play in Figure 3 illustrates the *intra-robot replanning* for T_b using Dribbling-Move. The robot changes its local position variable towards the passing location. As previously stated, the condition $Open$ is only being enabled implicitly as T_b is not attempting to find a new or better passing location. The condition will be enabled when the global planner declares the current pass probability above γ given some assumptions and simplification based on the current state. Dribbling-Move gives a relatively simple solution to enabling conditions $HasBall$ and $Open$, but it performs poorly in our example domain.

$HasBall$ can be defined as an **independent condition**. The ability to enable it is on a single robot's own ability to keep the ball or get the ball. Independent conditions can be enabled by changes in the robot's assigned variables to compensate for changes in the environment. $Open$ can be defined as a **dependent condition**. Its ability to be enabled is highly dependent on the other teammate(s) and/or opponent(s). Dependent conditions can be enabled by changes in the variables of the robots that are involved with enabling the condition. Obviously, the opponents' variables cannot be manipulated directly so it can be difficult to enable a dependent condition.

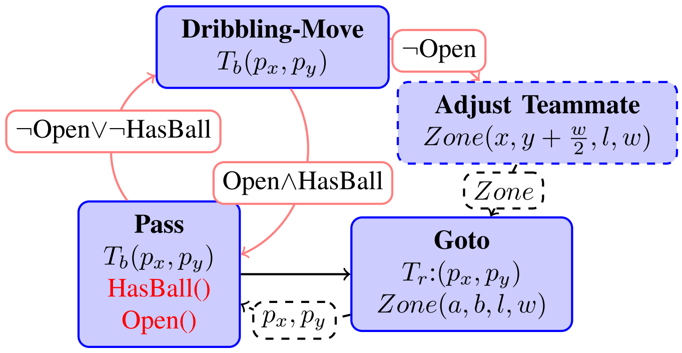

Fig. 4: *Intra-robot replanning* to more explicitly enable $Open$ by using the action Adjust Teammate to change the zone variable.

To enable $Open$, T_b would need to change its variables – this occurs through dribbling – and the variable(s) of the teammate's Tactic involved in the pass. Here we add the action **Adjust Teammate** to Dribbling-Move which changes the $Zone$ variable of the receiver's Goto Tactic, shown in Figure 4. Adjust Teammate is used if $Open$ is not enabled when dribbling. It places the zone in front of T_b and towards T_r as shown in Figure 5. A new location is found within the new zone for the pass and given to T_b and T_r. T_b can now explicitly attempt to enable the $Open$ condition by moving the teammate to a zone better situated for its own dribbling abilities.

The $Zone$ becomes a sliding window towards the right based on T_b's location, shown in Figure 5. This ensures that T_b is always close to the passing $Zone$ and the global planner finds T_r a new pass location.

IV. Experimental Results

A. Experimental Setup

We use the *passing with marking* domain as we described in our example with a field size of 9m long by 6m wide. T_b starts at (2m, 3m) with the ball directly in front of it at (1.86m, 3m) and opponent O_b behind it at (2.4m, 3m). Teammate T_r starts at (7m, 3m) with the opponent O_r in front of it at (6m, 3m). The Play is always the same with T_b passing to T_r, however the passing location changes quickly as the global planner attempts to find the best position within T_r's defined zone. The zone is defined as $(x = 4.5m, y = 0m, l = 4.5m, w = -6m)$, so the zone covers half of the field.

The pass must happen within one minute or it is considered a failed pass. If the ball goes outside field boundaries it is also considered a failed pass. Touching is defined as being within a robot radius plus a ball radius to the center

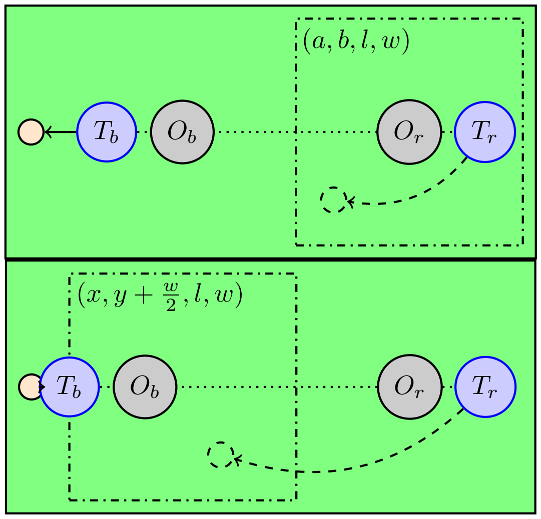

Fig. 5: Demonstrating the *Zone* adjustment, resulting in a change in pass location, from the plan in Figure 4.

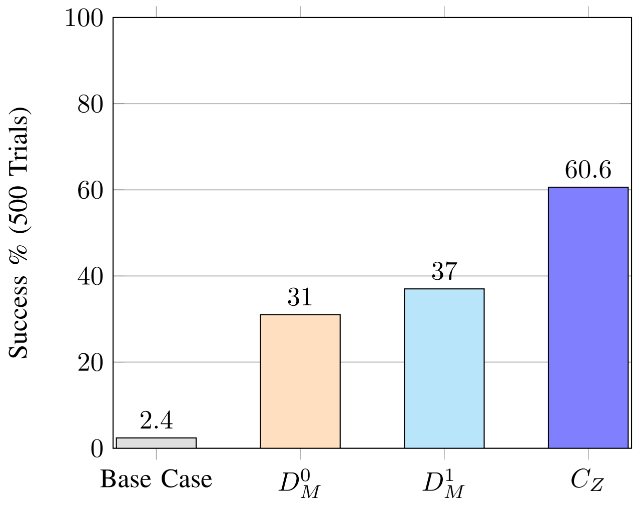

Fig. 6: Results for four different algorithms used by T_b in the *passing with marking* domain.

of a robot. If the ball is touching an opponent for five video frames then it has been intercepted and the pass has failed. A pass succeeds if T_r is touching the ball for five video frames.

The opponents place themselves in the direct line of sight between the two T_i. O_b maintains a close distance of three robot radii (3*0.09m) from T_b. O_r maintains a distance of 0.5m from T_r. When the ball is kicked, i.e., its velocity goes above 0.9m/s and it is two robot diameters away from T_b, O_r attempts to intercept the ball.

T_r uses the pass ahead algorithm to receive the ball as previously referenced. We are introducing **Change-Zone** which follows Figure 4. The zone parameters were determined by trial and error for improving the performance of passing. We compare **Change-Zone** to three algorithms for T_b with varying degrees of replanning to enable conditions:

- **Base Case**: Positions to face the ball and pass location, then immediately kicks as planned (no replanning).
- **Dribbling-Move, D_M^0, $\gamma = 0$**: Previously referenced algorithm with γ set for any pass.
- **Dribbling-Move, D_M^1, $\gamma = 0.15$**: Only passes if the pass probability is above γ.
- **Change-Zone, C_Z**: Uses D_M^1 with the Adjust Teammate action to enable the condition *Open* by altering the Teammate's Zone using: $(l = 2m, w = 6m)$.

B. *Results*

Each different T_b algorithm was run five hundred times in a physics-based simulator starting with the same formation of teammates, opponents, and the ball. The results can be found in Figure 6. Base Case fails almost every time and its few successes can be attributed to random chance as the ball is often stolen or blocked. Both Dribbling-Move algorithms improved the chance of a successful pass by enabling the condition $HasBall$, but passes were often intercepted by O_r. With the higher γ, Dribbling-Move further increased the success rate by waiting for γ to improve (even if just slightly as 0.15 is still very low). Change-Zone further increased the success rate by more explicitly attempting to enable $Open$ by moving T_r closer to find an open pass.

We used the binomial proportion test to verify that the different success rates of the algorithms were not due to randomness or limited sampling. The binomial proportion test is, as defined in [12]: given a set of N_1 observations in a variable X_1 and a set of N_2 observations in a variable X_2, we can compute a normal approximation test that the two proportions are equal, or alternatively, that the difference of the two proportions is equal to 0. A standard approach is that a p value of less than 0.05 can reject the null hypothesis. In our case, the null hypothesis defines that there is not enough evidence to show a statistically significant difference between the results of the algorithms.

As shown in Figure 7 each improvement on the Base algorithm is statistically significant when compared to previous algorithms. The initial change, D_M^0, occurred when replanning enabled the condition $HasBall$ by dribbling. Our contribution, C_Z, shows further significant improvement by replanning to enable the condition $Open$ by moving the teammate into a more beneficial zone. Our data demonstrate the large improvement when individual robots within a team can replan to enable all the conditions needed by their Tactics.

The authors of the Dribbling-Move algorithm showed the improved success rate of passing when marked by an opponent similar to O_b. With the inclusion of the second opponent, O_r, the success rate dropped dramatically. We can see that the improvement of Dribbling-Move mainly contributed to the enabling of the condition $HasBall$ and its ability to handle the close opponent, O_b. Changing-Zone's improvement was in changing the teammate's zone variable to benefit the dribbling algorithm's abilities. It brought the passing location closer for tighter, shorter passes and this helped improve the probability of enabling $Open$.

Binomial Proportion Test (Two-Tailed)		
Algorithm	**Algorithm**	**P Value**
Base Case	D_M^0	< 0.0001
D_M^0	D_M^1	0.0452
D_M^1	C_Z	< 0.0001

Fig. 7: Testing for the null hypothesis that the two proportions are equal and the difference is due to randomness.

V. Choosing a Replanning Algorithm

In this section, we describe how to choose which algorithms to use during the execution of the robot's Tactic. First, we describe an approach using the *worst case* analysis for choosing the appropriate algorithm. Second, we describe a *state based* approach that learns which algorithm to use given a world state. This approach allows the robot to use simpler algorithms when possible, like the base case, while learning when to use the more complicated ones like C_Z.

A. Worst Case

The task of our robot is to pass the ball to another teammate. This involves releasing the ball and losing all further control over it. It is an irreversible action, which cannot be replanned or halted. With irreversible actions, we would like to choose an algorithm that would minimize the worst case failure rate of the robot, because most of the failures cause the other team to obtain the ball.

Given the individual robot must pick to execute one of the algorithms in order to accomplish the task of passing, we define a method of choosing an algorithm based on the worst case. A standard approach for choosing one algorithm over another is if the algorithm is better in the worst case independently of the worst case actually occurring. Let $Z = \{0, 1\}$, where 0 is a failure and 1 is a success, and p be the function that gives the probability $p(z)$, where $\sum_{z \in Z} p(z) = 1$, for obtaining the failure or success in Z. We then assign a value $v(z)$ to the values of Z. We set $v(z)$ to be the total number of success for the algorithm, in Figure 6. We use the definition in [13],

$$p \succsim q \text{ if } min\{v(z)|p(z) > 0\} \geq min\{v(z)|q(z) > 0\} \quad (1)$$

Therefore, the algorithms would be ordered $C_Z \succsim D_M^1 \succsim D_M^0 \succsim BaseCase$. So, the **Change-Zone** algorithm would have the largest minimum value and be chosen as the default algorithm.

B. State Based

The worst case method will choose the algorithm that minimizes the worst case scenario, but by definition it does not consider the actual probability of the worst case happening. This can be seen as an issue if the worst case rarely, if ever, happens. Another possible issue is that each of the replanning algorithms alters the team plan causing changes that may have been unnecessary. In a normal game, situations like our example domain may not occur with high probability. In other words, if we have an open pass to our teammate with no opponents nearby there would be no point in dribbling the ball and/or moving the teammate closer.

A better method for picking the replanning method would then be based on the state of the world. Using the state information, it is possible to learn the likelihood of that algorithm being successful. Then we can choose the algorithm given its likelihood of succeeding. We must reiterate that, because the domain is dynamic and adversarial, success is very probabilistic even in the exact same state. Our robots move with some randomness which means that predicting the likelihood of an algorithm being successful at any given state will be probabilistic. The robots will likely diverge quickly into very different states in the future as subtle differences and randomness influence their trajectories.

We use neural networks to learn a function from world state to the probability of success. The state is described in the reference frame of the robot using the algorithm in order to help generalize different situations across the field.

1) Input State:

- T_b: (X, Y) position and velocity of T_b.
- **Ball**: (X, Y) position and velocity of ball in robot's frame of reference.
- **Teammates**: (X, Y) position and velocity of each teammate in robot's frame of reference.
- **Opponents**: (X, Y) position and velocity of each opponent in robot's frame of reference.

2) Output State:

- **Probability Value**: Likelihood of input state leading to a successful pass using an algorithm.

The data needed to train the neural network can be collected over many runs of each algorithm in various scenarios. We can simulate similar episodes in our example domain with variations on ball location, opponent's locations, and teammate's locations. As an episode starts we begin saving state information for each video frame. Then once the pass has either been received, intercepted, or gone out of bounds (the episode ends), we label all the saved states with a success value of 1 or a failure value of 0. As previously mentioned, even starting in the same state does not mean the robots will execute the same path in the future even when using the same algorithm. There is therefore a probabilistic nature to succeeding from a given state.

The neural network can then be trained using a supervised learning method. Given that similar, even the exact same, states will most likely be labeled success and failure, the network will actually learn the probability of succeeding within the training data. Therefore, the output is the likelihood of an algorithm succeeding from a given state.

Given that we train multiple neural networks, each trained on one algorithm, we will have the likelihood of success for each algorithm given a state. We then must decide on a method of choosing which algorithm to run based on the probability value. We use the expected utility of the algorithm as described in [13]:

$$p \succsim q \text{ if } \sum_{z \in Z} v(z)p(z) \geq \sum_{z \in Z} v(z)q(z) \quad (2)$$

Predicting Success or Failure of D_M^1			
Data Sets	**Accuracy**	**Recall**	**Precision**
Training Data	0.70	0.31	0.74
Next 200,000	0.65	0.26	0.65
Next 200,000	0.71	0.24	0.50

Fig. 8: Accuracy of predicting successes or failures for our example domain using the algorithm D_M^1.

where $v(z) = \frac{z}{C(a_i)}$ and $C(a_i)$ is an ordering cost for each algorithm, a_i.

3) Preliminary Experimental Evidence: To test the state based method, we used the open source library OpenANN [14]. The neural network has 20 inputs, see *Input State*, and 1 output as the probability of success, see *Output State*. We used a fully-connected network with three hidden layers with 100 neurons in each layer. We used LOGISTIC activators for the neurons and Mini-batch Stochastic Gradient Descent (MBSGD) to solve the supervised learning problem.

Our preliminary work has focused on determining the accuracy of predicting the success or failure of an algorithm using our example domain. We used the D_M^1 algorithm and repeatedly ran our example domain roughly five thousand times, gathering over 4 million states. We used two hundred thousand states to train the network over ten thousand iterations of MBSGD.

In Figure 8, we see that the accuracy of prediction is between 0.65-0.71. The low accuracy can be attributed to the randomness of a state leading to a success or failure because of the inherit randomness in the robot's performance. For example, the starting position of our *passing with marking* domain should be labeled as failure because on average it fails more often than it succeeds. This is confirmed as the start position produces the value 0.412. The neural network will always consider that position as a failure, which brings down the accuracy when the robot happens to succeed. In comparison to D_M^1's success rate in Figure 6, 0.37, the neural network has learned a similar approximation.

The low recall performance can be attributed to the large bias for failure in the dataset. The individual robot is started in an disadvantaged position and the likelihood of succeeding is already low. Similarly, when the robot actually succeeds in a given state it has probably failed multiple times from that exact same state in the dataset. The negative examples then highly outnumber the positive examples leading the neural network to perform poorly on recalling positive examples. A possible correction for this would be to balance the samples of positive and negative, however, this new bias would likely decrease the precision and accuracy of predicting if the algorithm will fail from a given state.

This preliminary work demonstrates the ability to learn the probability of success for a given state. Future work includes training neural networks for every algorithm and choosing which algorithm to execute based on their predictions and expected utilities.

VI. Conclusion and Future Work

We demonstrated an increase in team performance by providing the Tactic level with the ability to replan its robot's location, the ball's location, and a team member's variable in order to enable the Tactic's conditions. In our example, this was through dribbling and adjusting the preplanned zone of the team member. As the types of conditions were different, independent and dependent, enabling them required a difference in what variables needed to be changed. Replanning to enable both types of conditions clearly provides a better alternative to failing and waiting for the global planner to change its task.

We showed preliminary results on predicting the probability of succeeding for an algorithm. Future work has been left to picking the algorithm with the highest expected utility such that the robot succeeds often without defaulting to an unnecessarily complicated algorithm. Research is also needed to learn how to change variables in different situations. Specifically, it would be beneficial to learn the actual zone size and zone location that improves the pass success beyond the hard coded zone used in our experiments.

References

[1] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, December 2013.

[2] R. Simmons, T. Smith, M. B. Dias, D. Goldberg, D. Hershberger, A. Stentz, and R. Zlot, *A Layered Architecture for Coordination of Mobile Robots*. Dordrecht: Springer Netherlands, 2002, pp. 103–112.

[3] F. Pecora and A. Cesta, "Planning and Scheduling Ingredients for a Multi-Agent System," in *Proceedings of UK PLANSIG02 Workshop, Delft, The Netherlands*, 2002.

[4] B. Browning, J. Bruce, M. Bowling, and M. Veloso, "Stp: Skills, tactics, and plays for multi-robot control in adversarial environments," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 219, no. 1, pp. 33–52, 2005.

[5] S. Zickler, T. Laue, O. Birbach, M. Wongphati, and M. Veloso, "Ssl-vision: The shared vision system for the robocup small size league," in *RoboCup 2009: Robot Soccer World Cup XIII*. Springer, 2010, pp. 425–436.

[6] K. Talamadupula, D. Smith, W. Cushing, and S. Kambhampati, "A theory of intra-agent replanning," *ICAPS 2013 Workshop on Distributed and Multi-Agent Planning (DMAP)*, 2013. [Online]. Available: dmap13.pdf

[7] M. S. Atkin, D. L. Westbrook, and P. R. Cohen, "Capture the flag: Military simulation meets computer games," in *Proceedings of AAAI Spring Symposium Series on AI and Computer Games*, 1999, pp. 1–5.

[8] 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 the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 493–500.

[9] P. Stone, G. Kuhlmann, M. E. Taylor, and Y. Liu, "Keepaway soccer: From machine learning testbed to benchmark," in *RoboCup 2005: Robot Soccer World Cup IX*. Springer, 2006, pp. 93–105.

[10] J. P. Mendoza, J. Biswas, D. Zhu, P. Cooksey, R. Wang, S. Klee, and M. Veloso, "Selectively Reactive Coordination for a Team of Robot Soccer Champions," in *Proceedings of AAAI'16, the Thirtieth AAAI Conference on Artificial Intelligence*, Phoenix, AZ, February 2016.

[11] P. Cooksey, J. P. Mendoza, and M. Veloso, "Opponent-aware ball-manipulation skills for an autonomous soccer robot," in *Proceedings of the RoboCup Symposium*. Leipzig, Germany: Springer, July 2016, nominated for Best Paper Award.

[12] T. P. Ryan, *Modern Engineering Statistics*. Wiley, 2008, no. 124-126.

[13] A. Rubinstein, *Lecture Notes in Microeconomic Theory*. Dordrecht: SUNY-Oswego, Department of Economics, 2006, pp. 87–96.

[14] A. Fabisch, "Openann," 2017, https://github.com/OpenANN/OpenANN.