RESEARCH ARTICLE OPEN ACCESS

Towards a Versatile Opportunity Awareness Algorithm for Humanoid Soccer Robots using Time Petri nets

Seung-yun Kim, Daniel Ponsini, and Yilin Yang
(Department of Electrical and Computer Engineering, The College of New Jersey, USA)

----------**********----------

Abstract:

To keep up with the advancements in robotics, techniques for computing and modeling of behaviors must be developed and researched. Using Petri nets (PNs) to model complex systems allows for analysis and increased comprehension of the system itself. PNs can be used to show the logic behind a system or to create a visual representation of the steps a system will take. Although this alone can be useful, there are some areas in which normal PNs fail; in these cases introducing time into the nets can open modeling possibilities. Time can be used to literally represent the duration of an event or can be used to implement probability into a system. In this paper, Time Petri nets (TPNs) are used to model soccer playing robots whose movements are based on the proposed Selective Kick Opportunity Awareness Response (SKOAR) algorithm, which serves to guide the robots to the safest path to the goal. After modeling, simulation and testing, it was shown that the proposed algorithm outperformed both the Soccer Playing Allies Referencing Tract and Coordinate Underlay System (SPARTaCUS) algorithm and the Rapidly-exploring Random Tree (RRT) algorithm in goal score rate by about 3% and 7%, respectively and in goal attempt success rate by about 5% and 17%, respectively.

Keywords **— Modeling and Simulation, Petri nets, Time Petri nets, Robotics, Algorithms**

----------**********----------

I. INTRODUCTION

The field of robotics is expanding rapidly as many industries strive for automation. Robots are being used in many different areas as people attempt to streamline every process. Not only have they demonstrably exceeded human capabilities in speed and accuracy, they are often preferable in long term cost effectiveness as well. These trends have been reflected by innumerable growth in the past century as progress in this field continues to rapidly evolve; automated systems are increasingly adept at mimicking humanoid behaviors and can function with almost no assistance. With robots flourishing at every turn, the need for sophisticated modeling has become ever so important. However, the integration of robotics into everyday life is largely dependent on the autonomy of the robot, especially for humanoid models.

A humanoid robot is a sophisticated robot whose structure is based on the human body, most typically used in studying bipedal locomotion. Humanoid NAO robotic platforms [26], for example, have been used in various applications, from simple educational tools [3] to genetic algorithm development for fitness-based sitting pose optimization using predefined motions [2] and object recognition using a modified simultaneous recurrent network [1].

There are also many applications in a soccer robot environment. The Robot Soccer World Cup (RoboCup) is an annual international competition consisting of multiple leagues that promotes robotics and artificial intelligence (AI) research [27]. One such league, the Standard Platform League (SPL), uses up to 5 Aldebaran NAO robots [26] to play soccer games. Since success in these games is dependent in part on player mobilization and kick execution, extensive research has been conducted to study walking and kicking motions using robot subjects. There are two well adopted approaches for controlling biped robot's walking in a soccer robot environment: Central Pattern Generator (CPG) and Zero Moment Point (ZMP). Bavani et al., [5] implemented and simulated an optimized genetic algorithm using the CPG approach and Strom et al. [22] proposed omnidirectional walking using ZMP balance metrics. Further research simulated and tested an adaptive kicking algorithm based on visual feedback as opposed to the typical key frame technique, but it was limited to short distances [15]. Learning algorithms have also been explored as a means of improving a robot's ability to score penalty goals based on the kicking point, foot

trajectory, and effectiveness of previous kicks [7, 14]. Barrett et al. [4] proposed a strategy in which they prioritized the quickest pre-set kick motion available that would advance the ball towards the goal.

To keep up with the advancements in actual robotics, computing and modeling of behaviors must be developed and researched as well. Petri nets (PNs) are one of many well-known methods researchers use for the modeling and simulation of robot behaviors. PNs are a graphical and mathematical tool that can model a variety of multifaceted systems. Many researchers and scientists have been using PNs for years to help visualize the problems at hand. Kuo and Lin [13] used agent-oriented Petri nets to model methodology of controlling autonomous robots. However, this attempt was at the abstract level and failed to show the simulation results of modeling. Zouaghi et al. [24] introduced a generic hybrid monitoring approach for autonomous mobile robots using Petri nets. This technique allowed for the detection of inconsistencies using a model of environmental representations but implementation of the proposed modified Petri nets were constrained to modeling of the navigation process. Kim and Rew [9] proposed the limit cycle method for robotic navigation. Their proposed method differed from two different approaches: the deliberative and reactive approaches. These two forms of navigation suffered from high computational costs and difficulty of design, respectively. The limit cycle method was faster and more efficient but failed to account for avoiding obstacles.

The paper organization is as follows: Section 2 offers background concepts key to the understanding of the PNs, Time Petri nets (TPNs) and their modeling techniques; Section 3 shows the process of the development of the PN including robot soccer competitions, scenario, and algorithm; Section 4 explains how the algorithm was implemented in the PN and the results of the implementation of the algorithm; and Section 5 concludes the paper with a summary of the algorithm's performance.

II. BACKGROUND

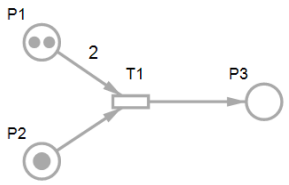
A. *Petri net Definitions*

A PN is a valuable tool in graphically and mathematically analyzing a wide variety of scenarios or systems. The general concept can be described as a tuple

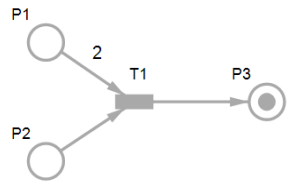
$$PN = (P, T, A, W, M_0)$$

where P is a finite set of places, i.e., P = {P0, P1, P2, …}, T is a finite set of transitions, i.e., T = {T0, T1, T2, …}, A is a finite set of arcs, i.e., $A \subseteq (P, T) \cup (T, P)$, W is a weight function that assigns a weight to all elements in A, i.e., W: A→ℕ, and M_i is a marking that assigns a whole number to every place representing the amount of tokens in that place after *i* firings, i.e., M_i: P→𝕎. The initial marking, M_0, is referenced in the description of a given PN, for example, in Fig. 1, M_0 = (2, 1, 0) and M_1 = (0, 0, 1). Finally, a set of all input places (or output places) for a given transition, T_n, is represented as $IP(T_n)$ (or $OP(T_n)$).

Pictorially, places are typically represented as circles and transitions as rectangles. Places are linked to transitions and transitions to other places through arcs that are represented simply as arrows. Arc weight is designated as a nonnegative integer value or assumed to equal 1 if not specified, for example, A(P1, T1) = 2, A(P2, T1) = 1, and A(T1, P3) = 1 as shown in Fig. 1. Places and transitions are often numbered or named after a step in a procedure. A PN fires markings known as tokens from place to place through the transitions connecting them to represent the progression of the procedure. Specifically, a token or number of tokens is fired by a transition from one place to another when it is enabled. A transition is enabled when all places leading to it are marked as true with a token [10, 17]. Typically, places represent conditions, transitions represent events, tokens represent status, and arcs transport tokens using rays as presented in Fig. 1.



(a) Before firing T1



(b) After firing T1

Fig. 1 Graphical representation for a Petri net

Transitions in PNs fire according to the firing rule [17]: a transition, T_n, is enabled if all P in $IP(T_n)$ have tokens equal to the weight of the arc leading to the transition; an enabled transition may or may not fire (depending on if the event occurs); the firing of an enabled transition removes a number of tokens equal to the input arcs' weights from each input place and adds a

number of tokens equal to the output arcs' weights to each place in $OP(T_n)$.

There are also two useful properties in a PN modeling: reachability and liveness. A PN is reachable from M_1 to M_n if and only if there is a firing sequence of transitions, $T_1, T_2, \ldots, T_{n-1}$, that transforms M_1 to M_n. A PN is live if and only if for every reachable state and every transition *t*, there is a state M_n reachable from M_m that enables *t* [17].

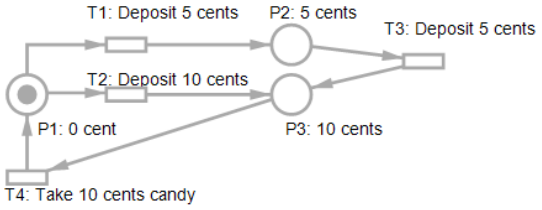
B. ***Petri net Modeling***

To maintain the best balance between clarity and complexity, the modeling and simulation program used should exercise the properties described above and implement them in an intuitive manner without sacrificing analytical capabilities. Two well-known Petri net modeling and simulation programs, HPetriSim [25] and TINA [28], are used in this paper.

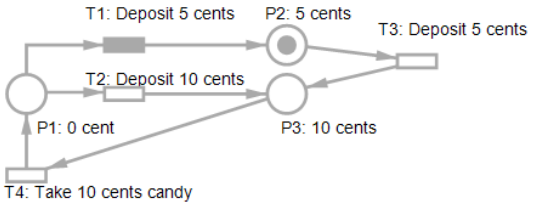
HPetriSim has a graphical editor for editing and simulating Petri nets. It provides many functions to analyze models including verifying underlying relationships among arcs, checking for the absence of deadlock states during execution (liveness), and ensuring the completion of the PN's execution (reachability). Its best attribute is simulation of system behavior, which shows tokens traveling from one place to another. One toolbox that has been developed to model TPNs is **TI**me petri **N**et **A**nalyzer (or **TINA**). The TINA toolbox includes many different kits that allow for different methods of TPN study, such as textual and graphical modeling tools, structural analysis tools, and step simulator tools. The NetDraw (nd) tool is a modeling tool that was used to graphically represent TPNs [28].

In the example shown in Fig. 2(a), the transition T1 is enabled only if the leading input place P1 is marked with tokens, indicting the conditions they represent are verified or true. If that is the case, T1 will empty the token of P1 and deposit tokens into the output place P2 equal to the weight of the arc between T1 and P2 via the firing process as shown in Fig. 2(b). This can be described as being unable to take candy from a vending machine unless a user deposits 10 cents (five cents twice or ten cents once), as shown in Fig. 2(c) (reachability). Places and transitions can link to form a circle, creating a cyclical process, or may end in various sinks. As long as the procedure makes logical sense, a Petri net can simulate a looping process for countless runs (soundness) [17].

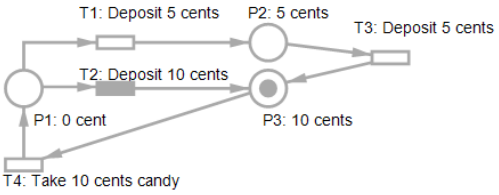
When developing a Petri net model, many additional features must be included to ensure consistency and proper functionality. For example, in Fig. 3(a), place P4 with an inhibitor arc is introduced to prevent depositing a penny. As the number of tokens in place P4 satisfies the weight of the inhibitor arc, it will disable the unwanted transitions. When P4 contains a token, transition T5 will always be inhibited, forcing deterministic and desired results.



(a) Initial setting



(b) Taking 5 cents first

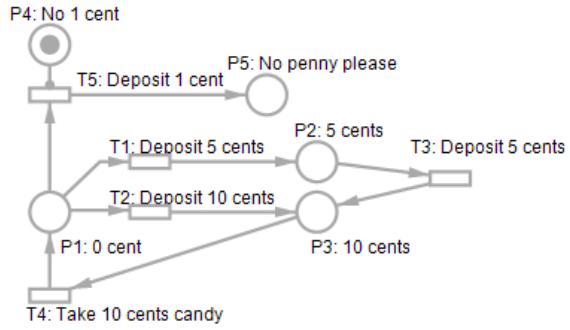


(c) Ready to take 10 cents candy

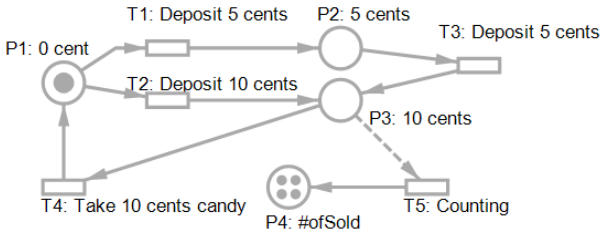
Fig. 2 An example of a Petri net created in the HPetriSim simulator

While inhibitor arcs are useful for restricting transitions, many cases exist where a transition should instead be enabled given certain conditions. This introduces the need for test arcs, which allow a transition to fire when the weight of the arc is met. Similar to the inhibitor arc, the test arc does not remove tokens from the originating place, a very useful property when attempting multiple comparisons or for preserving the tokens of previous states as demonstrated by Fig. 3(b). If A(P3, T5) were a standard arc, transitions T4 and T5 cannot both fire due to a deficiency in tokens, causing one to fire first in a nondeterministic nature. However, with the introduction of a test arc, this issue resolved by conserving the token upon firing A(P3, T5). By using

test arcs (illustrated as dotted rays) to resolve this issue, T5 can fire multiple times as shown in Fig. 3(b) to identify the total number of candies sold. Although the use of test and inhibitor arcs is necessary for the process of modeling the systems, they introduce the undesirable effect of remnant data (i.e., excess tokens). In most situations, the tokens must be cleared after use to assure the model functions as intended.



(a) Inhibitor Arcs



(b) Test Arcs

Fig. 3 Transition control through arc variants

Although PNs are a powerful tool on their own, many variants of PNs have spawned to attempt to diversify the situations they are able to model [20]. Petri nets can be conjoined with time mechanics, yielding TPNs.

C. ***Petri net Modeling***

The introduction of time into Petri net theory gives a way to many opportunities regarding system modeling. Since the real world applications incorporate time into all operations, therefore, a time variable is needed to properly model operations with Petri nets. Two different time-based Petri net models were introduced: Time Petri nets (TPNs) [16] and Timed Petri nets (TdPNs) [21]. The TPN is defined as a PN with time intervals, notated as the interval between two real numbers a and b where $a \leq b$, assigned to each transition whereas the TdPN includes a finite firing duration, as a single number, to each transition in order to timetable processing orders. Timing constraints can be also statically associated with different timing locations: places [6], arcs [8] and transitions [18]. A TPN is formally described as a 6-tuple

$$TPN = \{P, T, A, W, Z_0, I\}$$

where P is a finite set of places, T is a finite set of transitions, A is a finite set of arcs, W is a weight function, and Z_0 is the initial marking. I is a tuple $[a_i,b_i]$ such that for all I: $a_i, b_i \in \mathbb{N}$ and $a_i \leq b_i$. I is the interval of the TPN where a_i is the earliest firing time and b_i is the latest firing time [16]. A simple TPN model using TINA [28] modeling tool is shown in Fig. 4 to show these properties. T1 can fire at 1 time unit at the earliest and is guaranteed to fire by 4 time unit. T2 can fire as soon as it is enabled and is guaranteed to fire 5 time units after it gets enabled. T3 can fire 1 time unit after it becomes enabled at the earliest and is guaranteed to fire 4 time units later.

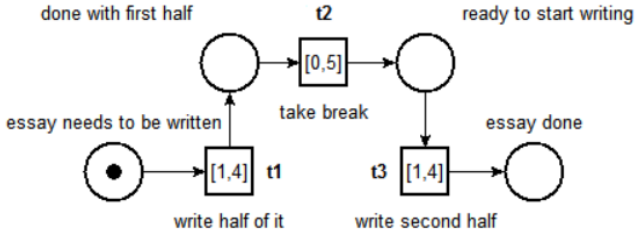


Fig. 4 Graphical model of TPN

Along with the addition of time intervals, a new firing rule is developed to fit the new logic of the PNs. If the input conditions (IP) for a given transition T_i hold for a time $t \geq a_i$, then T_i can fire. If the IP for a given transition T_i holds for a time $t = b_i$, then T_i will fire. With these additional stipulations, systems with literal time constraints or prioritized actions can be modeled using TPNs [16].

In order to clarify the use of time intervals, an example of a gas station model is developed using the TINA modeling and simulation tool as shown in Fig. 5. In this model, time intervals are assigned to each transition to represent the real time interval of different events. The concept modeled in the TPN is based around a gas station with one worker and two pumps. At any given time, there can only be one car at each pump, so the place representing a pump inhibits the arrival of another car to that same pump. The two transitions, "car arrives 1" and "car arrives 2", are assigned a time interval of *[0,20]*, which means that as long as there is no car currently at the pump, a car may show up immediately but will be guaranteed to show up after 20

time units have passed. Once a car has arrived at a pump, the worker should approach the car if one is available (transition "worker arrives 1" or "worker arrives 2"). The time interval *[1,1]* means that it will always take the worker one time unit to get to the car. Once the worker has arrived and there is a car present, they will begin to fill up the tank. This event is assigned the time interval *[3,5]*. This accounts for the varying size and fullness of the gas tanks. This task can thus take as little as 3 time units but will not take more than 5 time units. Once a car's tank is full, the car can leave and the worker becomes available again.

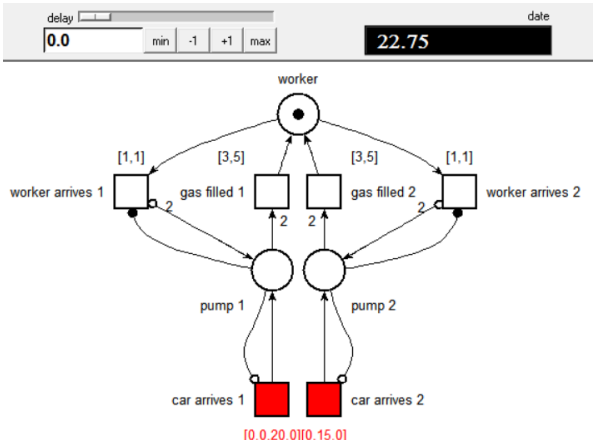
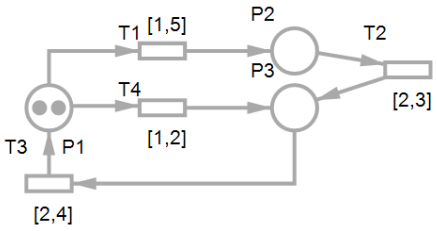
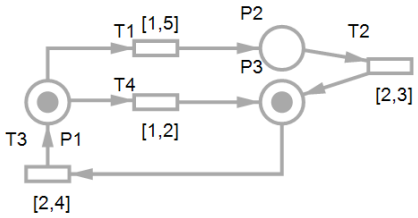


Fig. 5 An example of TPN: gas station model



(a) Markings for Z_0



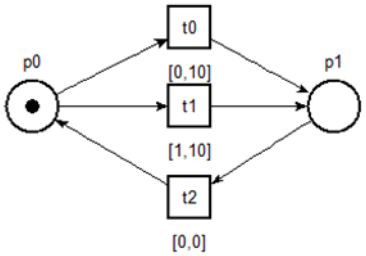
(b) Markings for Z_2

Fig. 6 An example for p-marking and t-marking

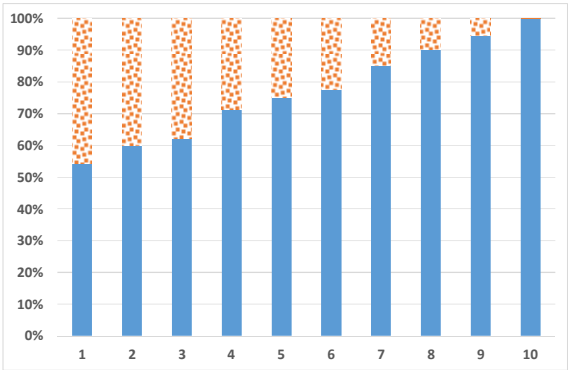
Adding a time component to a PN changes the meaning of a marking. In addition to the number of tokens on each place as discussed in Section 2.1, now called p-marking, it is necessary to consider each enabled transition with the amount of time that has passed since its last enabling and each disabled transition with the symbol #, called t-marking [19]. A TPN's marking (Z_0) can be described as a pair with a p-marking (M_0) and a t-marking (H_0): $Z_0 = (M_0, H_0)$. Based on time elapsing many TPN markings can be generated, for example, initial markings at 0 time unit are $M_0 = (2, 0, 0)$ and $H_0 = (0, \#, \#, 0)$, and markings at 1 time unit can be $M_1 = (2, 0, 0)$ and $H_1 = (1, \#, \#, 1)$ as shown in Fig. 6(a). In the time unit is 2, markings can be $M_2 = (1, 0, 1)$ and $H_2 = (2, \#, 0, 0)$ as shown in Fig. 6(b).

The time intervals can be related to probabilities of firing sequences of a TPN, for example, in Fig. 6(b), T1, T3 and T4 are enabled. T1 has updated interval as *[0,3]* and T4 resets interval after firing to *[1,2]* again. One of them will fire within 2 time units but T1 has a higher firing probability because T1 is enabled before T4. This is an import factor for time intervals so that the prioritization of firing sequences can be implemented.

D. ***Probabilistic Analysis on Time Petri nets (TPNs)***



(a) TPN model for analyzing TFP



(b) Results of testing

Fig. 7 Testing and analyzing transition firing probabilities

The relationship between time intervals in TPN models and probability of a transition firing is investigated. Different TPN models using TINA are

established in order to analyze the firing rates and time intervals, e.g., a model shown in Fig. 7(a). Data about the probability of a transition firing is collected based on over 900 iterations of TPN simulations. Fig. 7(a) illustrates a TPN model with the time interval of T_1 varying in each TPN. An equation is developed to determine the probability of a transition firing given a TPN with time intervals assigned to each transition. Fig. 7(b) shows the results of simulations and the probability of T0 depicted as the solid region and T1 shown as the dotted region.

The Transition Firing Probability (TFP) equation is then developed to reflect the data that are collected. Let P_j be a place with a token in it. Let each transition T_i (where $T_i \in z_j$) have a firing interval of $[a_i,b_i]$ and a_i , $b_i \in \mathbb{Z}$. Let a_i be the minimum time required before T_i is enabled and b_i be the maximum time until T_i will fire, i.e., $0 \leq a_i \leq b_i$. Let z_j be the set of mutually exclusive transitions such that all transitions with at least one common place P_j in their IP where the firing of any one of the transitions in the set will disable all other transitions. Also, let y_j be the minimum of all b_i in z_j and s_j be the minimum of all a_i in z_j. Finally, if T_i is enabled, N_{k-1} will be the number of transitions enabled in z_j at time (k-1) and if transition T_i is not enabled, then $N_{k-1} = \infty$. Then,

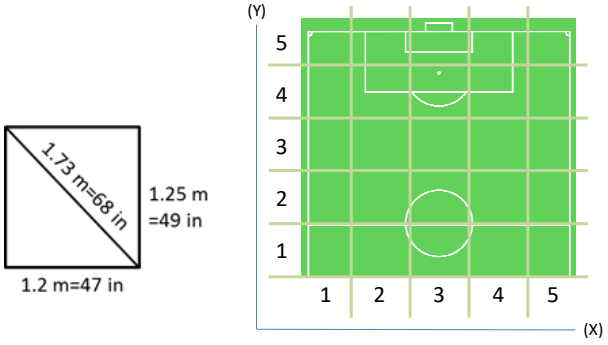
$$\text{TFP}\,(t_i) = \sum_{k=s_i+1}^{y_i} \frac{1}{y_i - s_i}\left(\frac{100}{N_{k-1}}\right) \qquad (1)$$

III. DEVELOPMENT OF ALGORITHMS

In order to demonstrate the powerful modeling capabilities of TPNs, a scenario is created based on RoboCup [27], an annual international robotics competition with an emphasis on autonomous soccer playing robots. Specifically, the regulations for the RoboCup Standard Platform League (SPL) are used as a basis to define scenario parameters, including the use of NAO robots, developed by Aldebaran [26], as subjects. The system is designed to be a realistic recreation of a RoboCup SPL match consisting of two teams of five robots each. Simulations are conducted with the objective of guiding one team of robots, henceforth referred to as *Allies*, to pass a ball across a field in an attempt to score a goal against the opposing team, henceforth referred to as *Opponents*.

A. *Soccer Field Coordination and Kicking Behavior Analysis*

Field specifications used in RoboCup SPL are described as 6*m* by 9*m*. Modeling of a game of robot soccer necessitates the field to be quantified in a way understandable to the robots and easily communicable for both human planners and information exchange. The field should not, however, be modeled with such a degree of precision as to create an unnecessarily complex representation of the game. By partitioning the field into 1.2*m* by 1.25*m* sections, a feasible and accurate depiction can be achieved. This level of division allows for a full field resolution of 5 by 8 when using the individual cells shown in Fig. 8(a). Trials measuring the distance covered by an SPL 2015 standard ball [26] when kicked by a NAO robot suggested the average length reachable to be approximately 1.78*m*, a distance slightly longer than the diagonal of the individual grid size proposed (1.73*m*). Based on these measurements, it is reasonably probable the result of a kick will shift the ball to an adjacent coordinate, no matter where the robot is in a particular cell. Behavior of the allies is assumed to be on the offensive, i.e., starting with possession of the ball and attempting to score a goal. Conversely, behavior of the opponents is assumed to be defensive, i.e., attempting to repel the allies' offense. Based on these conditions, the majority of the allies' half of the field is not relevant for consideration as having the ball deep within allied territory implies defensive play. This allows for the truncation of the 5 by 8 field to a 5 by 5 field shown in Fig. 8(b), decreasing the complexity of the model while retaining all necessary information.



(a) Individual cell size (b) Coordinate system

Fig. 8 Grid size and coordinate system with field for reference

Since NAO robots have curved feet, when kicking the ball, the angle of trajectory can vary. In order to account for this, it was necessary to determine the angular range of deviation of the ball when kicked by a robot. To do so, simulations were conducted, and kick data were recorded and analyzed [18]. The result of the study identified the safe zone, i.e., 60° range in front of the robot, based on the vision span of the NAO robot [26]. Ball trajectories outside the safe zone would be

considered a miskick. From these simulations, it was determined that the robot kicked the ball into the safe zone 80% of the time, i.e., safe kicks [18].

B. ***Previously Proposed Algorithms***

Many algorithms have been developed to optimize the path. Two different algorithms were selected to compare the proposed algorithm in this section. One well-known algorithm for robot path finding is the Rapidly-exploring Random Tree (RRT) algorithm [12, 18] as shown in Fig. 9. The RRT algorithm offers a time constrained set of controls to move from a starting location X_{init} to a desired destination X_{goal}. Finding an optimized path depends on the validity of the state space model being used [11]. Furthermore, if the robots were to roam the soccer field randomly, several challenges arise, such as advancement of the ball without a robot present or the advancement of the robots while the ball is left behind. This can be regulated by a control input α. An algorithm should seek to quantify the vast variables that a robot would face in such scenario. As a result, the algorithm used in this paper is designed within a specific system and set of parameters, similar to those featured in RoboCup competitions [27]. In robot soccer competitions such as RoboCup, kicking the ball toward the intended direction is a key factor for successful teams. Thus, executing efficient algorithms to receive and pass the ball is imperative.

RRT Algorithm

1. Select an initial node X_{init}.
2. Choose a random state X_{rand}.
3. Pick the node X_{near} that is nearest to X_{rand} using the control input α.
4. Add the node X_{near} as X_{new}.
5. Repeat steps 2 to 4 until X_{goal} is included.
6. Find the complete path from X_{init} to X_{goal}.

Fig. 9 RRT Algorithm [18]

The Soccer Playing Allies Referencing Tract and Coordinate Underlay System (SPARTaCUS) algorithm was proposed with the four key principles: forward movement, side movement, passing, and opponent avoidance as elaborated in Fig. 10 [23]. The SPARTaCUS algorithm is an iterative process where each robot is aware of the positions of the ball, opponents, and fellow teammates. While the robots move about the playing field, SPARTaCUS takes effect when one robot secures the soccer ball and begins locating allies and identifying optimal passing partners. The first order of the algorithm, $(x_b, y_b) \neq (x_{oj}, y_{oj})$, is a safety check, ensuring that opponents are not present near the ball. If this check fails, the first order is immediately terminated and SPARTaCUS instead issues the second order, $(x_b, y_b) = (x_b, y_b)$, to defend the current position. Once the first or second order completes successfully, SPARTaCUS concludes and the algorithm reiterates [23].

SPARTaCUS Algorithm

1. If $(x_b, y_b) \neq (x_{oj}, y_{oj})$
   - I. If $y_b = 5$
     - a) If $x_b = 1$
       - i. If $(x_{ai}, y_{ai}) = (2, 5)$, then $(x_b, y_b) = (2, 5)$
       - ii. Else $(x_b, y_b) = (x_b, y_b)$
     - b) If $x_b = 5$
       - i. If $(x_{ai}, y_{ai}) = (4, 5)$, then $(x_b, y_b) = (4, 5)$
       - ii. Else $(x_b, y_b) = (x_b, y_b)$
     - c) If $(x_b = 2) \vee (x_b = 3) \vee (x_b = 4)$, then attempt goal shot
   - II. If $|\alpha| = 0$, then $(x_b, y_b) = (x_b, y_b)$
   - III. If $|\alpha| = 1$, then $(x_b, y_b) = \alpha$
   - IV. If $|\alpha| = 2$
     - a) Else If $(3, y_{ai}) \in \alpha$, then $(x_b, y_b) = (3, y_{ai})$
     - b) If $(2, y_{ai}) \in \alpha) \wedge ((4, y_{ai}) \in \alpha)$, then $(x_b, y_b) = (x_b, y_{ai})$
     - c) Else If $(1, y_{ai}) \in \alpha$, then $(x_b, y_b) = (1, y_{ai})$
     - d) Else If $(5, y_{ai}) \in \alpha$, then $(x_b, y_b) = (5, y_{ai})$
2. Else $(x_b, y_b) = (x_b, y_b)$

Fig. 10 SPARTaCUS Algorithm [23]

C. ***Proposed Algorithm***

Common strategies allocate up to three robots for offensive play while a fourth robot is stationed as a goalie [23]. Responsibility of the fifth robot tends to vary depending on algorithm implemented, though many institutions choose to designate it as a defensive player in imitation of human soccer team formations. For the purposes of the proposed algorithm, we assume identical team builds for the allies and opponents. Each team consists of three offensive-oriented robots possessing dynamic freedom of movement. Their range of operation on the playing field encompasses all coordinates described in the system. Each team also consists of one defensive-oriented robot and one goalie robot, both of which possess static freedom of movement. Their range of operation is confined to a single coordinate position to protect. For all robots, current position and possible new positions are identified by coordinates. Fig. 11 illustrates robot maneuvering using the allies as an example, where places represent locales and tokens represent current robot location. When advancing forward, each ally may have a maximum of 3 possible new coordinate positions to relocate to. These moves may include moving directly forward, where the x coordinate is unchanged, moving diagonally left, where

the x coordinate is decreased by 1, and moving diagonally right, where the x coordinate is increased by 1. In all cases, the y coordinate is increased by 1 to reflect the advancing motion.

Allies are free to move to any of these new positions so long as they are physically possible. Physically possible entails, for example, that an ally cannot move further left if they are in the leftmost column (1, y) or warp to the rightmost column (5, y) from the leftmost column. If by circumstance no advancing move is ideal, allies may move horizontally along the x-axis to a new position in the hopes that there will be better options available.

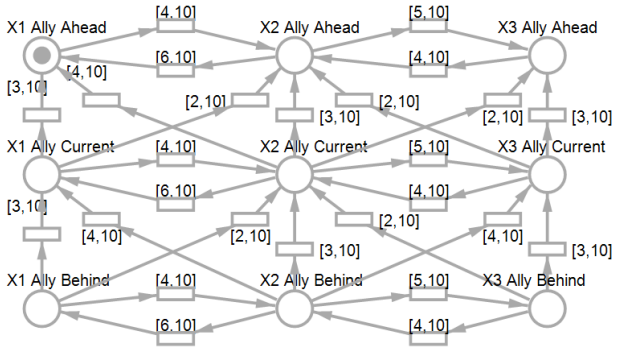


Fig. 11 Representation of ally movement module

The ideal movement pattern for the robots attempts to funnel them into the center of the field since the robots will have the best chance of scoring a goal while in X3 (i.e., the middle of the field). If they cannot reach the center, their next best option is to move to X2 or X4, directly adjacent to the center. Their lowest priority while still advancing forward is to move to the outermost columns of the field (X1 and X5). If no forward motion is available, the priority scheme is repeated for sideways movement (with the highest in the center and lowest at the edges). The small scale model of the ally robot motion shown in Fig. 11 has these priorities implemented using the TFP equation, eq. (1). For the robots on the very edge of the field, there are only three options for movement and for the robots in the middle of the field, there are five movement options, as previously described. Thus, there needed to be two separate sets of transitions with different probabilities. For example, a robot in the current row in the first column, it will be available to move ahead to X2, ahead to X1, and sideways to X2, with priorities highest to lowest respectively. By assigning a lower start time to a certain transition in a set of mutually exclusive transitions, it will have a higher priority based on the TFP equation, eq. (1). Table 1 shows the time intervals that were assigned to replicate the priority and the probability was calculated using the TFP equation, eq. (1) for both sets of transitions.

The only other restriction in terms of movement is the current location of the ball. At any given moment, the ball should be under control of an ally. Whichever particular robot that happens to be will not venture beyond its current coordinate position. Instead, its objective will be to pass the ball to an adjacent ally ahead. Adjacency in this case is defined as either the same column or a neighboring column. Once the receiving ally has control of the ball, its objective, too, will be to pass the ball on ahead. In addition, allies are cautious not to advance too far ahead of or fall too far behind the ball and are no further than 1 row away from it at any given time. Similarly, opponents are interested in securing the ball for themselves and seek to remain no further than 1 row away from it. Assuming they are effectively pursuing their strategic interests, it is to be expected that all offensive players can be found within a limited bound of three rows centered on the given location of the ball, as seen in Fig. 12, making coordinated formations a critical factor for success.

TABLE 1 TRANSITION FIRING PROBABILITY OF TRANSITIONS

From X1 Current	To	X2 Current	X1 Ahead	X2 Ahead		
	[1]T.I.	*[4,10]*	*[3,10]*	*[2,10]*		
	[2]Pr.	25	31.25	43.75		
From X2 Current	To	X1 Current	X3 Current	X1 Ahead	X2 Ahead	X3 Ahead
	[1]T.I.	*[6,10]*	*[5,10]*	*[4,10]*	*[3,10]*	*[2,10]*
	[2]Pr.	10	13.13	17.29	23.54	36.04
[1]T.I.: Time interval given in the transition and [2]Pr.: Probability in %						

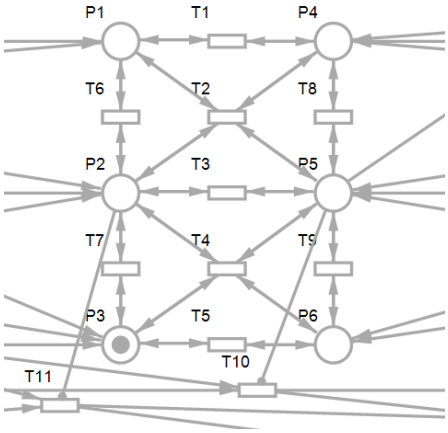


Fig. 12 Representation of dynamic opponent movement module

Fig. 13 describes in more detail the mechanics of the robots passing behavior. Ball passes are regulated to prioritize moves that generate favorable outcomes or are more likely to generate favorable outcomes. In a situation where multiple teammates are open to pass the ball to, the ally currently in control of the ball will prefer to pass to the teammates closer to the middle of the field, i.e., in decreasing order of favorability, $x = 3$, $x = 2$ or $x = 4$, and $x = 1$ or $x = 5$. The rationale for this stems from the probability that a robot near the center field will likely have more opportunities to pass the ball than a robot at the outskirts of the field and have higher success rates when attempting a goal shot given that the goal posts aligns with the central column in Fig. 8(b).

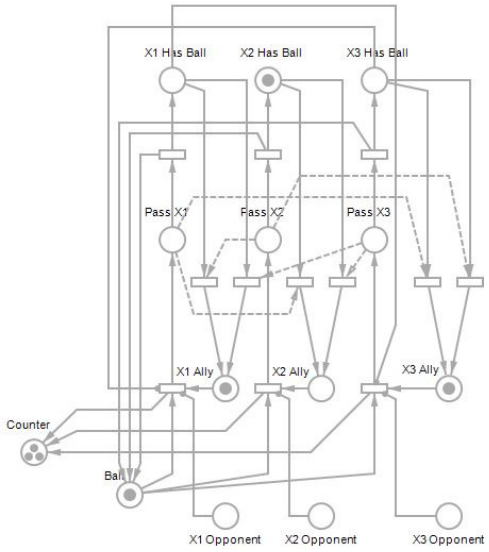


Fig. 13 Representation of ball passing module

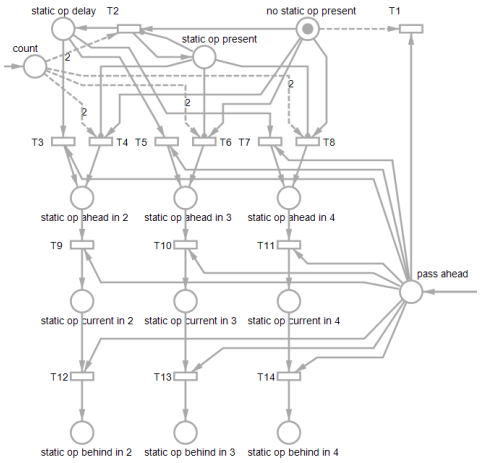


Fig. 14 Representation of static opponent cell assignment module

The greatest threat and most unpredictable variable in any soccer game is the action of the opposing team. As shown in Fig. 13, passing is influenced not only by ally formations, but opponent presence as well, as shown in Fig. 14. The greatest advantage and most crucial objective is to retain control of the ball. Opponent robots detected within the same coordinate position as the ball significantly raise the risk of interception. Therefore, in the event the ally currently possessing the ball is in close proximity of an opponent, control of the ball becomes contested. Allied passing actions are inhibited and effort for both teams is refocused on securing the ball. Contested control is resolved by whichever team is able to first reinforce the contested cell with a supporting teammate, as shown in Fig. 15. The proposed algorithm operates based on two assumptions; (a) that the allies start the game with control of the ball, and (b) do not lose long-term control of the ball. Therefore, if the opponents are able to successfully contest the ball and subvert control, the premise of operation is lost and ground for failure.

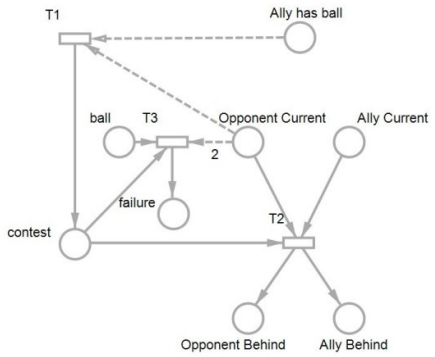


Fig. 15 Representation of contested ball module

The three key principles of allied movement, passing, and opponent avoidance are what compose the Selective Kick Opportunity Awareness Response (SKOAR) algorithm, which is further elaborated in Fig. 16. SKOAR can be described as a tuple consisting of the following notation:

$$((x, y)_b, (x, y)_{ai}, (x, y)_{oj}, \alpha)$$

where $(x, y)_b$ is the coordinate of the ball, $(x, y)_a^i$ is the coordinate of offensive-oriented ally robots with $i = \{1, 2, 3\}$ to differentiate between specific teammates, $(x, y)_{oj}$ is the coordinate of opponents with $j = \{1, 2, 3, 4\}$ to differentiate between specific opponents, and α is the set of all adjacent allies defined by $\{(x, y)_{ai} \mid (x, y)_{ai} = (x + \varphi, y + \psi)_b \wedge (x, y)_{ai} \neq (x, y)_{oj}\}$ with $\varphi = \{-1, 0, 1\}$ and $\psi = \{0, 1\}$. Since the allied defender and goalie possess only static movement, they are not present in the system

described in Fig. 8(b) and therefore truncated from the set *i*. This condition also bars the opponent goalie from the set *j*.

SKOAR is the culmination of the behaviors described by Fig. 11 through 15 where each ally robot is monitoring the current location of the ball and passing candidates of α. This algorithm is designed to run whenever the robot is in possession of the ball. If the robot is not in possession of the ball, it is moving, therefore these rules do not apply to their decision making. The algorithm first checks if control of the ball can be contested through $(x, y)_b \neq (x, y)_{oj}$. If the risk for a contest is deemed too high, SKOAR deprioritizes candidate searching and instead instructs $(x, y)_b = (x, y)_b$ to cease further transport of the ball. Assuming the contested check does pass, the algorithm then compiles ally positions. If an ally's position matches the definition of adjacency, their coordinates are included in set α. Depending on the cardinality of α, there may be one, multiple, or no candidates to receive a pass. If |α| is 0 or 1, there is no choice to make; the only possibility is to defend the position or pass the ball to the only open teammate, respectively. Should there be multiple candidates to pass to, SKOAR evaluates their position and prioritizes teammates farther ahead and closer to midfield. If all candidates are equally viable near midfield, preference goes to the ally to the right of the ball.

SKOAR Algorithm

If $(x, y)_b \neq (x, y)_{oj}$
- I. If $y_b = 5$
  - a) If $(x_b \neq 2) \vee (x_b \neq 3) \vee (x_b \neq 4)$
    - i. If $(2, 5)_{ai} \in \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
    - ii. If $(4, 5)_{ai} \in \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
  - b) If $(x_b = 2) \vee (x_b = 3) \vee (x_b = 4)$, then attempt goal shot
- II. If $|\alpha| = 2$
  - a) If $(3, y_b + 1)_{ai} \in \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
  - b) Else If $((2, y_b + 1)_{ai} \in \alpha) \vee ((4, y_b + 1)_{ai} \in \alpha)$
    - i. If $(2, y_b + 1)_{ai} \notin \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
    - ii. If $(4, y_b + 1)_{ai} \notin \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
    - iii. If $(2, y_b + 1)_{ai} \in \alpha \wedge (4, y_b + 1)_{ai} \in \alpha$, then $(x, y)_b = (4, y_{ai})_b$
  - c) Else If $(1, y_b + 1)_{ai} \in \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
  - d) Else If $(5, y_b + 1)_{ai} \in \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
  - e) Else If $x_b \neq 1$
    - i. If $(3, y_b)_{ai} \in \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
    - ii. Else If $((2, y_b)_{ai} \in \alpha) \vee ((4, y_b)_{ai} \in \alpha)$
      - A) If $(2, y_b)_{ai} \notin \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
      - B) If $(4, y_b)_{ai} \notin \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
      - C) If $(2, y_b)_{ai} \in \alpha \wedge (4, y_b)_{ai} \in \alpha$, then $(x, y)_b = (4, y_{ai})_b$
    - iii. Else If $(1, y_b + 1)_{ai} \in \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
    - iv. Else If $(5, y_b + 1)_{ai} \in \alpha$, then $(x, y)_b = (x_{ai}, y_{ai})_b$
- III. If $|\alpha| = 1$, then $(x, y)_b = \alpha$

Else $(x, y)_b = (x, y)_b \wedge y_{ai} = y_b$

Fig. 16 Selective Kick Opportunity Awareness Response Algorithm

A special case occurs if $y_b = 5$, meaning the ball has been escorted to very end of the opponent's half of the field. In this scenario, there is the opportunity to attempt a goal shot. Rather than check for adjacency, allies concern themselves with setting up the goal shot. $X_b = 3$ presents the best possibility for a straight trajectory to the opponent goal, therefore the allies will attempt to move the ball as close as possible to this coordinate and execute the kick. Once the first or second order completes successfully, SKOAR concludes and the algorithm reiterates.

IV. MODELING AND SIMULATION OF ALGORITHMS

Comparative studies of the SKOAR algorithm against contemporary alternatives were facilitated through TPN simulations. A TINA adaptation for every module described by Fig. 11 through 15 was constructed and assembled into a singular stepwise representation of a RoboCup SPL match. The Ball Passing module serves as the reference frame for simulation duration; for every pass forward made, a counter is incremented to indicate progress made toward the final objective of a goal shot. Ball Passing consults Ally Movement and Dynamic Opponent Movement before outputting a decision, shifting control of the ball to another robot. The process is cyclical and continues for a specified number of iterations, based on the length of the soccer field to be crossed. For these simulations, the system described in Fig. 8(b) is assumed as the environment for the game thereby establishing a maximum of 4 iterations. At the antepenultimate iteration, the Static Opponent Cell Assignment module is triggered to generate an opponent defender robot. Since defender robots are typically designated to protect the general vicinity of the penalty area, their maximum spawn range is much more constrained such that the opponent defender will not be first encountered by the allies until they near the goal. Fig. 17 illustrates the possible regions of the soccer field in which specific robot types may first appear.

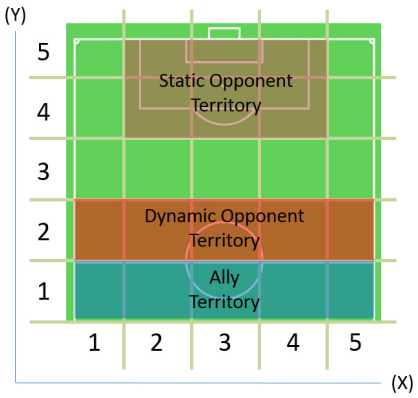


Fig. 17 Possible starting positions for players

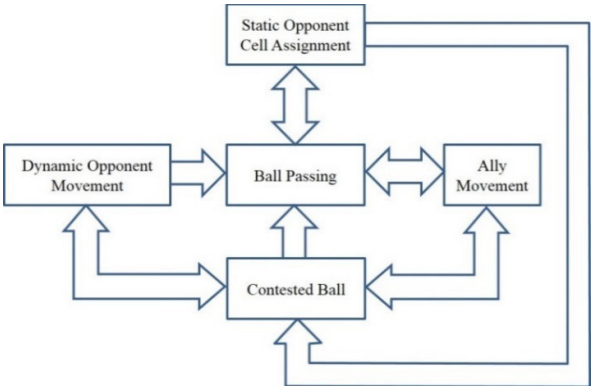


Fig. 18 System diagram of RoboCup SPL game simulation

At any point, the Contested Ball module may be activated if the conditions for the scenario arise. In such a situation, Contested Ball takes precedent over Ball Passing and is in effect until either Ally Movement or Dynamic Opponent Movement yield an output that resolves the conflict. If the contest is resolved in favor of the opponents, allied control of the ball is lost and the trial is deemed unsuccessful as the goal was not reached. If the contest is resolved in favor of the allies, Ball Passing reassumes control and the simulation continues. Once the number of iterations has reached the maximum value, the endgame can be initiated. The success of a goal shot conducted by an ally robot is dependent on the final position of the allies. Additionally, the orientation of the ball and player relative to the goal has an observable impact on the accuracy of a kick. The most advantageous position is directly across the goal, centered at $(3, 5)_b$ in Fig. 8(b). Favorability decreases symmetrically the farther the ball is from this origin. The time intervals associated with goal shot success are specified based on these criteria as well as the findings of Ponsini et al. [18] which found physical limitations of robot kick accuracy. The final outcome of a simulation can either result in a goal scored, a goal missed, or an interception by the opponents preventing the goal from being reached. The structure of module interaction can be seen in Fig. 18.

Based on these parameters, performance of SKOAR, SPARTaCUS, and RRT were evaluated over the course of 200 simulations per algorithm for a total sample size of 600 simulated matches. The decisions made by the algorithms in response to a given set of factors may differ, necessitating the need for three variations of the system described by Fig. 18.

TABLE 2 SIMULATION RESULTS

SKOAR		
Goal Scored	Goal Missed	Goal Not Reached
52	27	121
SPARTaCUS		
Goal Scored	Goal Missed	Goal Not Reached
46	30	124
RRT		
Goal Scored	Goal Missed	Goal Not Reached
37	38	125

For each model, the time intervals controlling the outcomes of specific modules were adapted to replicate a response as similar as possible to a specific algorithm's methodologies and intended action. In all simulations run, initial conditions consisted of starting positions for ally robots specified at $(2, 1)_{a1}$, $(3, 1)_{a2}$, $(4, 1)_{a3}$, and starting position for the ball at $(3, 1)_b$. Dynamic Opponents are stationed in the row directly across. With 3 dynamic opponents and 5 possible columns to assign them in, 10 distinct opponent formations exist. In the interest of observing different ally reactions, each formation is tested for 20 out of the total of 200 simulations. The outcome of these trials is listed in Table 2.

Simulations conclude when a token is deposited into a place identifying the state of the system. From this, it is possible to extrapolate quantitative characteristics of algorithm performance. Success is dependent on the capability of a team to pass a ball, avoid opponents, and score a goal. As a result, the algorithms are evaluated granularly based on their ability to demonstrate competency in these skills. Three particular statistics were observed: (a) goal score rate, (b) goal attempt rate, and (c) attempt success rate. Due to the possibility of the opponents successfully contesting the ball, not every simulation ends in a score attempt. In real world games

of soccer, control of the ball is frequently exchanged between opposing teams. Thus, there are many cases where the allies are interrupted and fail to reach the end of the field. It can be observed from Table 2 that SKOAR, SPARTaCUS, and RRT are all able to navigate past the opponents at an approximately equal rate of 40%, 38%, and 38%, respectively. However, in cases where allies are able to approach the goal, positioning is a defining factor to the success of a goal shot. In this aspect, SKOAR demonstrates a notable improvement over competing algorithms, shown by Fig. 19. As a result of more favorable positioning, SKOAR has been shown to improve overall score rate over SPARTaCUS and RRT by 3% and 7%, respectively.

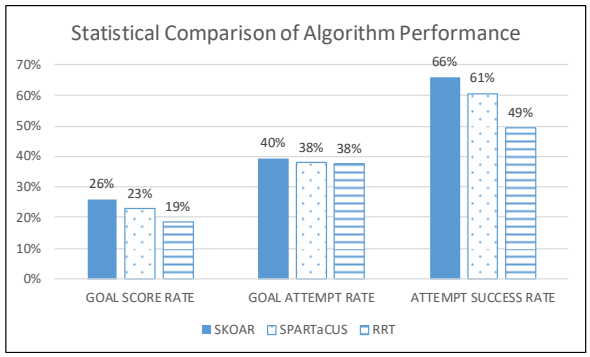


Fig. 19 Analysis of SKOAR performance compared to competing algorithms

V. CONCLUSIONS

In this paper, an alternative to existing robot soccer algorithms was proposed utilizing the unique modeling capabilities of Petri nets. Petri nets are a developing field which allow for the simulation and analysis of complex systems. Specifically, Time Petri nets were explored as a useful tool for modeling and simulation of robotic navigation of a soccer field. The SKOAR algorithm proposed enhances a robot's ability to avoid opposing robots and coordinate with friendly robots to position themselves into strategically favorable formations. The accommodations for more realistic factors such as contested spaces in the SKOAR algorithm allows for more efficient planning.

The TPN developed takes advantage of the different properties of PNs available in the TINA toolbox, including inhibitor and test arcs, and was able to model accurate simulations of soccer games. Using the data gathered from the TPN, it was illustrated how algorithm performance could be quantitatively studied and compared. From these results, it was shown that SKOAR yielded a higher goal score rate; 26% compared to 23% for SPARTaCUS and 19% for RRT. Further work will seek to verify these results in real world tests using physical robot platforms.

ACKNOWLEDGMENT

The work was partially supported by the New Jersey Space Grant Consortium, and the summer MUSE program at The College of New Jersey.

REFERENCES

[1] M. Alam, L. Vidyaratne, T. Wash, K. Iftekharuddin, "Deep SRN for Robust Object Recognition: A Case Study with NAO Humanoid Robot," in *Proc. of the IEEE South East Conference, SoutheastCon2016*, pp. 1 – 7.

[2] M. Al-Hami, R. Lakaemper, "Sitting Pose Generation Using Genetic Algorithm for NAO Humanoid Robots," in *Proc. of the IEEE Workshop on Advanced Robotics and Its Social Impacts,* 2014, pp. 137 – 142.

[3] A. Alkhalifah, B. Alsalman, D. Alnuhait, O. Meldah, S. Aloud, H. Al-Khalif, H. Al-Otaibi, "Using NAO Humanoid Robot in Kindergarten: A Proposed System;" in *Proc. of the IEEE International Conference on Advanced Learning Technologies*, 2015, pp. 166 – 167.

[4] S. Barrett, K. Genter, T. Hester, M. Quinlan, P. Stone, "Controlled Kicking under Uncertainty," in *Proc. of the 5th Workshop on Humanoid Soccer Robots*, HSR-10, 2010.

[5] A. Bavani, H. Ahmadi, H. Nasrinpour, "A Closed-Loop Central Pattern Generator Approach to Control NAO Humanoid Robots' Walking," in *Proc. of the IEEE International Conference on Control, Instrumentation and Automation*, 2011, pp. 1036 – 1041.

[6] J. Coolahan, N. Roussopoulous, "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets," *IEEE Transactions of Software Engineering*, SE-9, pp. 603 – 616, 1983.

[7] T. Hester, M. Quinlan, P. Stone, "Generalized Model Learning for Reinforcement Learning on a Humanoid Robot," in *Proc. of the IEEE International Conf. on Robotics and Automation*, 2010, pp. 2369 – 2374.

[8] L. Jacobsen, M. Jacobsen, M. Moller, J. Srba, "Verification of Timed-Arc Petri nets," *Theory and Practice of Computer Science*, 6543, pp. 46 – 72, 2011.

[9] D. Kim, K. Rew, "Second-order Nonlinear Function Navigation Method for Fast Mobile Robots," in *Proc. of the International Conference on Control, Automation, and Systems*, 2008, pp. 2836 – 2840.

[10] S-y. Kim, "Modeling and Analysis of a Web-based Collaborative Enterprise using Petri nets," in *Proc. of the 2008 IEEE International Conference on Information Reuse and Integration*, 2008, pp. 422 – 428.

[11] M. Kothari, D. Gu, I. Postlethwaite, "An Intelligent Suboptimal Path Planning Algorithm Using Rapidly-exploring Random Trees," in *Proc. of the European Control Conference*, 2009, pp. 677 – 682.

[12] J. Kuffner, S. LaValle, "RRT-connect: An Efficient Approach to Single-query Path Planning," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2000, pp. 995 – 1001.

[13] C. Kuo, I. Lin, "Modeling and Control of Autonomous Soccer Robots Using Distributed Agent Oriented Petri nets," in *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, 2006, pp. 4090 – 4095.

[14] X. Li, Z. Liang, H. Feng, "Kicking Motion Planning of Nao Robots Based on CMA-ES," in *Proc. of the 27th Chinese Control and Decision Conference*, 2015, pp. 6158 – 6161.

[15] H. Mellmann, Y. Xu, "Adaptive Motion Control with Visual Feedback for a Humanoid Robot," in *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, 2010, pp. 3169 – 3174.

[16] P. Merlin, D. Farber, "Recoverability of Communication Protocols-Implications of a Theoretical Study," *IEEE Transactions on Communications*, 24(9), pp. 1036 – 1043, 1976.

[17] T. Murata, "Petri Nets: Properties, Analysis and Applications," in Proc. of the IEEE, **77**, pp. 541 – 574, 1989.

[18] D. Ponsini, Y. Yang, S-y. Kim, "Analysis of Soccer Robot Behaviors using Time Petri nets," in *Proc. of the IEEE International Conference on Information Reuse and Integration*, pp. 270 – 274 (2016)

[19] L. Popova-Zeugmann, *Time and Petri nets*, Springer, 2013.

[20] H. Rakkay, H. Boucheneb, O. Roux, "Time Arc Petri Nets and their analysis," in *Proc. of the 9th Conference of Application of Concurrency to System Design*, 2009, pp. 138 – 147.
[21] C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. Massachusetts Institute of Technology," Technical Report 120, 1974.
[22] J. Strom, G. Slavov, E. Chown, "Omnidirectional Walking using ZMP and Preview Control for the NAO Humanoid Robot," *RoboCup 2009*, J. Baltes et al. (Eds.), Springer-Verlag Publisher, pp. 378 – 389, 2010
[23] Y. Yang, D. Ponsini, S-y. Kim, "Ball Control and Position Planning Algorithms for Soccer Robots using Fuzzy Petri nets," in *Proc. of the ISCA International Conference on Computers and Their Applications*, 2016, pp. 387 – 392.
[24] L. Zouaghi, A. Alexopoulos, A. Wagner, E. Badreddin, "Mission-based Online Generation of Probabilistic Monitoring Models for Mobile Robot Navigation using Petri nets," *Robotics and Autonomous Systems*, vol. **62**, pp. 61 – 67, 2014.
[25] HPetriSim, HPetriSim Petri nets Tools Database, Available at: https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db/hpsim.html or https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html. Accessed 2 April 2017
[26] NAO Robot Documentation, Aldebaran, Available at: https://www.aldebaran.com/en. Accessed 2 April 2017
[27] RoboCup International, RoboCup Standard Platform League, Available at: http://www.tzi.de/spl/bin/view/Website/WebHome. and http://www.informatik.uni-bremen.de/spl/bin/view/Website/WebHome. Accessed 2 April 2017
[28] TINA, Time Petri Net Analyzer, Available at: http://projects.laas.fr/tina//. Accessed 2 April 2017