EPiC Series in Computing

Volume 69, 2020, Pages 317–326

Proceedings of 35th International Conference on Computers and Their Applications

Soccer Playing Robot Goal Scoring Algorithm Using Fuzzy Petri nets

Paul Brodhead, Garrett Hope, and Seung-yun Kim
Department of Electrical and Computer Engineering
The College of New Jersey
Ewing, NJ 08628, USA
{brodhep1, ghope1, kims}@tcnj.edu

Abstract

To facilitate the creation of a robotic soccer team, a robust kicking strategy and algorithm must be developed. Through the use of Fuzzy Petri nets, a strategy was made and developed into an algorithm to produce a 95% success rate. Image processing and recognition was used to implement this algorithm onto NAO robots.

1 Introduction

Creating a fully autonomous team of robot soccer players has been a goal of the robotics community for a long time. The RoboCup Federation has made it it's mission to create a team of robot soccer players that can beat a world class team [11]. To further this goal, it is necessary to have a robust and effective strategy for players to score goals. Players on the field must have a way to effectively send a ball into the opponent's goal while it is being defended by a goalie. Petri nets (PNs) are an effective way to get to this stage. A PN is a type of directed graph that is typically used in modelling systems, algorithms, and workflows. These systems can then be described as a list of "if-then" statements for implementation into any number of real-world situations. This along with the ability to simulate the effectiveness of a system makes PNs extremely useful and versatile in algorithm development.

Many researchers have been using PNs to help visualize the problems, ranging from modeling methodology of controlling autonomous robots [6], as well as to implement an optimal object avoidance approach for autonomous mobile robots [8].

This paper also focuses on a subset of PNs called Fuzzy Petri Nets (FPNs). An FPN combines the nature of uncertainty found in fuzzy logic with the graphical nature of Petri nets [3], allowing for more powerful algorithm development and analysis tool. This ability to incorporate uncertainty provides a more appropriate methodology to the development of artificial intelligence. For example, FPNs were used to develop an algorithm for efficient obstacle avoidance techniques using a memory module [2].

The algorithm development process is described in the following sections: Section 2 details the background of Fuzzy Petri nets, NAO Robots, the RoboCup, and Image Processing; Section 3 goes through the methodology of building the new algorithm, along with its implementation on real NAO

robots; Section 4 takes the results of the comparison and determines the viability of the new algorithm developed using a FPN; Section 5 then summarizes the findings of this paper in a conclusion marking the end of the paper.

2 Background

The ability for a robot to perform complex tasks autonomously and without the need for human intervention is currently the bleeding edge of robotics research. An increasingly popular method of modeling these tasks and their execution is through the use of PN models, as they are an intuitive way to show the workflow – and consequently the steps of the algorithm – through the graphical format of a PN. This format lends the PN and its subset the FPN to be a very effective form of developing and testing artificial intelligence programs and applications [5]. While robotics remains heavily entrenched in practical application, theoretical and mathematical developments in algorithms designed to improve human-robot interaction are still useful; however, purely theoretical development is not of much use until backed up with proper real-world testing. For this, the NAO robot makes an excellent choice as one of the competition categories in the RoboCup is with all NAO soccer robots [12], and the proven platform allows all focus to go towards developing and implementing the algorithm.

2.1 Petri net Model

Petri nets are a mathematical tool that are a type of directed graph describing a finite state automata. They are an excellent tool in building and simulating systems because of the intuitive descriptions of workflows provided by the Petri net. The standard PN is a graph consisting of two types of nodes: places and transitions; these nodes represent the states and steps of an algorithm. Tokens then move through this graph as described by the rules of a PN [7]. A Petri net is formally defined as a 5-tuple:

$$N = (P, T, F, W, M_0) \quad (1)$$

The 5-tuple N is made up of P, a finite set of places:

$$P = \{p_1, p_2, \ldots p_k\} \quad (2)$$

T, a finite set of transitions:

$$T = \{t_1, t_2, \ldots t_m\} \quad (3)$$

F, a finite set of arcs:

$$F = \{f_1, f_2, \ldots f_n\} \quad (4)$$

W, a weight function that defines how many tokens a given arc consumes and produces, and M_0, the initial marking that defines how many tokens are in each place at the starting state of the PN. M_0 is a type of marking, M, which is the distribution of tokens in each place throughout the Petri net, which can generally be defined as:

$$M : P \rightarrow \mathbb{N} \quad (5)$$

In order for Petri nets to be considered a type of graph, the nodes must be connected by some type of edge. This is accomplished through the use of arcs, which are a type of directed edge which makes the PN a directed graph. Arcs can only connect places and transitions with each other, not with themselves. A transition can only fire when all the places connected to it by arcs have enough tokens to satisfy the weights of said arcs, and after the transition fires, an appropriate number of tokens will be deposited in each of the places leading from the transition. Therefore, it can be summarized that tokens represent the execution of an algorithm, places represent the states of an algorithm, transitions represent the steps in an algorithm, and arcs represent the connections between steps and states.

These rules are demonstrated by the example Petri net of Figure 1. Here, there is a token in the place *p1* which represents the initial marking M_0. This token satisfies the weight of the arc going into transition

t1 (weight of 1), allowing *t1* to fire. This causes the token in *p1* to be consumed by *t1*, and then *t1* will produce a token which will deposit a token in place *p2*.

Figure 1: Example of a Petri net with transition *t1* firing

2.2 Fuzzy Petri net Model

The concept of Fuzziness was developed in order to allow for a mathematical representation of uncertainty in logic problems, in a similar fashion to how people deal with uncertainty in decision making by using vague boundaries as to whether something will happen or not. An excellent description of how fuzziness works is provided by L.A. Zadeh with the phrase "computing with words" [10].

The ability to work with uncertainty is accomplished through the use of a membership function that describes how much a certain element belongs to a set. The membership function is defined as:

$$\mu_x : X \rightarrow [0,1] \quad (6)$$

Equation 6 maps each element in a set to a value between 0 and 1. This value is called the degree of membership and allows for the inclusion of certain elements into a set if they are arbitrarily close to fully being a member of the set. Elements with a degree of membership closer to 0 are those that do not belong to the set, and elements with a degree of membership closer to 1 are those that do belong to the set [9].

The ability to deal with uncertainty in mathematical situations provided by Fuzzy logic can be combined with the existing Petri net frameworks to create what is called a Fuzzy Petri net [8]. The main difference between a standard PN and a FPN is the fact that rather than having discrete requirements for a transition to fire, FPNs allow for fuzziness in the ability for a transition to fire. Formally, a Fuzzy Petri net is defined as an 8-tuple:

$$N_{Fuzzy} = (P, T, F, W, M_0, \alpha, \gamma, \sigma) \quad (7)$$

In this case, *P*, *T*, *F*, *W*, and M_0 are all defined in the same manner as in a standard Petri net, where Equations 2, 3, and 4 represent *P*, *T*, and *F* respectively; and *W* and M_0 are the weight function and initial marking, respectively. α, γ, and σ are the new additions to the Fuzzy Petri net [7]. α is the certainty value associated with each token in the net, having a value between 0 and 1. The certainty value is defined as:

$$\alpha : P \rightarrow [0,1] \quad (8)$$

γ is the threshold value associated with each transition in the net, having a value between 0 and 1. The threshold value is defined as:

$$\gamma : T \rightarrow [0,1] \quad (9)$$

σ is the truth value, which is determined as a result of the certainty and threshold values of the token and transition being fired. The truth value is defined as:

$$\sigma = \alpha \times \gamma, \quad \alpha \geq \gamma \quad (10)$$

In order to determine if a transition will be fired, the certainty value of the tokens leading into the transition are compared to the threshold value of the transition. If the certainty value of the token is greater than or equal to the threshold value of the transition, then the transition is allowed to fire. Once the transition has been fired, the new token being produced will have a certainty value equal to that of the truth value calculated from the firing. If the certainty value of the incoming token is less than the threshold value of the transition, then that transition is not able to fire.

In order to demonstrate these properties of FPNs, an example FPN can be made from a fuzzy implication. In this case, the transition would represent how true the implication is. The example FPN can represent the fuzzy implication "if there are many clouds in the sky, then it will rain". In order to create the FPN, the certainty and threshold values must be set for the token and transition, respectively.

For this net, let $\alpha = 0.95$ (representing a sky nearly completely covered in clouds) and $\gamma = 0.90$; this setup will allow the example FPN to fire the transition, which is shown in Figure 2.

Because $\alpha \geq \gamma$, transition *t1* is allowed to fire, consuming the token in place *p1* and depositing a token in place *p2*. The new token being deposited in *p2* will have a certainty value equal to the truth value of the fired transition, σ. σ is calculated by multiplying together α and γ, yielding a result of 0.95 * 0.90 = 0.855. Therefore, the token in *p2* has a certainty value of 0.855, meaning that it is very certain that it will rain.

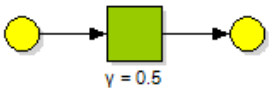

Figure 2: Example of Fuzzy Petri net

Humanoid NAO robotic platforms have been used in many different applications, from simple educational tools [1] to object recognition using a input image to predict the distance of an object [4]. NAOs can be programmed through their NAOqi SDK, or a GUI software called Choregraph. For this paper, the Python 2.7 NAOqi SDK is used along with the OpenCV library for image processing. With this combination, the NAO robot can be communicated with and controlled by vision and motion proxies, enabling instructions for where to kick the ball to be sent, based on what the NAO is "seeing". The RoboCup Federation is an organization dedicated to the advancement of robotics and artificial intelligence. The main way they do so is by hosting competitions in different leagues of robot types. The Standard Platform league is comprised of teams made up entirely of Nao robots [4]. Like in regular soccer, different robots fulfill the roles of regular players and goalie. The OpenCV Python library is used to determine multiple factors necessary for the developed algorithm to be implemented. These consist of the goal location, the goalie NAO location, and the kicking NAO location. The image processing done consists of a combination of contour and color detections.

3 Methodology

When creating the kicking strategy, a number of factors were taken into account. First, the positions of both the player and goalie are considered. Then an ideal location for the ball to be kicked to is calculated. From this, the projected ball location and known goalie position are compared to check the likelihood that the ball makes it into the goal. To have a useful idea of where the player is, a constant distance from the goal is assumed. This is possible because the kicking strategy will activate as soon as the player is within kicking range, which is a constant distance measured from the goal. The arc covered by possible shooting positions is then split into four equally spaced segments and labelled 00, 01, 10, and 11 from left to right facing the goal. Whichever segment of the arc the player is at the time of shooting is the position from which the ideal ball location is calculated.

The goalie position is determined in a similar fashion. The goal is split up into four equally sized spaces, labelled 00, 01, 10, and 11 from left to right as seen by the player. Whichever quadrant the goalie is standing in is considered to be the position of the goalie. A diagram of the layout is shown in Figure 3.

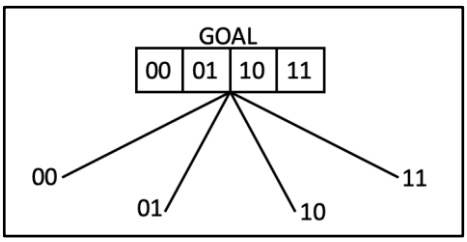

Figure 3: Field Layout

The reasoning behind having four positions for the goalie and player to be in is for ease of use in implementation on electronics. Four positions each means that they can be represented by two bits each with no loss of efficiency. This allows for a quicker calculation of the algorithm in any computing scheme. The algorithm can be described in both PN form and in the form of a series of if-then statements. The proposed algorithm and the implementation of the algorithm as shown in Figure 4 (a) and (b), respectively. The completed PN can be seen in Figure 5.

```
if (goaliePosition = 11):
          AND (playerPosition == 00):
                    kickBallTo = 00
else
          KickBallTo = playerPosition-01
if (goaliePosition < 11):
         AND (goaliePosition = playerPosition):
                   kickBallTo = playerPosition+01
else
         KickBallTo = playerPosition
```

(a) Proposed Algorithm

```
if (goaliePosition == 00):
    if (playerPosition == 00):
        kickBallTo = 01
    if (playerPosition == 01):
        kickBallTo = 10
    if (playerPosition == 10):
        kickBallTo = 10
    if (playerPosition == 11):
        kickBallTo = 11
if (goaliePosition == 01):
    if (playerPosition == 00):
        kickBallTo = 00
    if (playerPosition == 01):
        kickBallTo = 00
    if (playerPosition == 10):
        kickBallTo = 10
    if (playerPosition == 11):
        kickBallTo = 11
if (goaliePosition == 10):
    if (playerPosition == 00):
        kickBallTo = 00
    if (playerPosition == 01):
        kickBallTo = 00
    if (playerPosition == 10):
        kickBallTo = 11
    if (playerPosition == 11):
        kickBallTo = 11
if (goaliePosition == 11):
    if (playerPosition == 00):
        kickBallTo = 00
    if (playerPosition == 01):
        kickBallTo = 00
    if (playerPosition == 10):
        kickBallTo = 10
    if (playerPosition == 11):
        kickBallTo = 10
```

(b) Implementation of the algorithm

Figure 4: If-Then statements describing the kicking algorithm

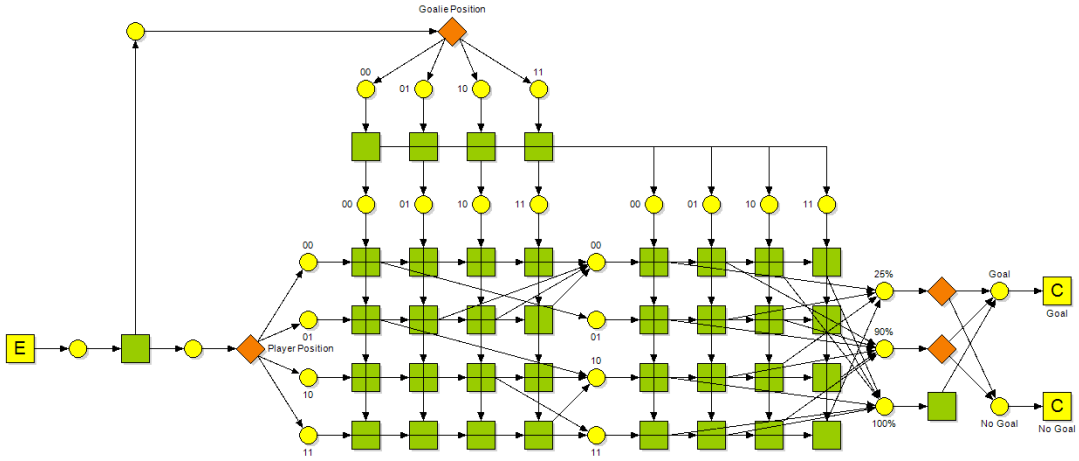

Figure 5. The fully realized Petri net of the algorithm

The positions to which the ball is kicked are determined based on a combination of where the goalie is and where the closest position in the goal is relative to the player. Because the goalie robot is not very capable of moving to intercept the ball, kicking to the quadrant directly next to the goalie is not ruled out as it still has a high chance of getting in. The percentage chances of a ball scoring a goal is based off of observations from multiple years of RoboCup competitions, where the goalie had a hard time blocking balls from anywhere besides directly in front. The FPN for picking which quadrant the goalie is in and which angle the player is kicking from are shown in Figure 6. Each transition in both FPNs has a threshold value of 0.5, and the certainty values are given to the tokens when first assessing the situation the player is in.

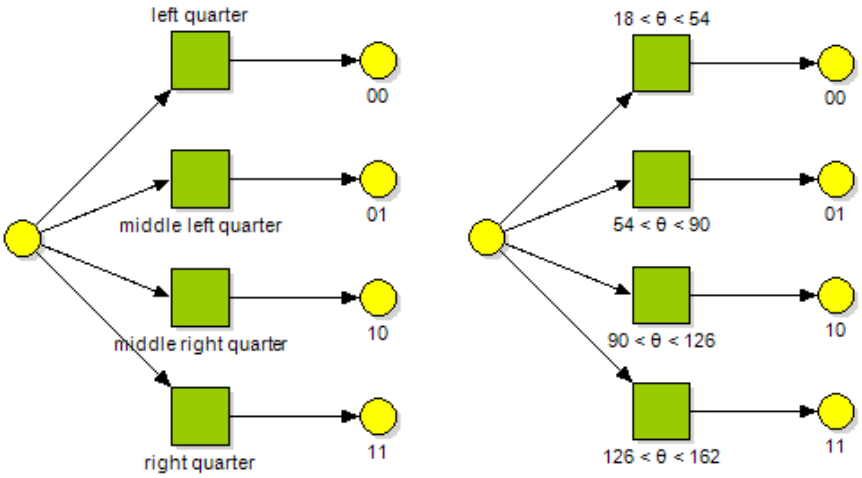

Figure 6: FPNs for determining where the robots are

To implement the algorithm, an image stream must be established between the NAO robot, and the computer running the Python script. The “ALVideoDevice” proxy is used to collect the captured images from the NAO, and then they are converted to OpenCV images, so the library can be used to process them. The unprocessed image of the NAO’s view is shown in Figure 7.

The first step in processing the image is to determine the location of the goal. This is done by detecting the contours of the image and filtering out the contours that do not meet the size and shape criteria of the goal. The results of this step can be seen in Figure 8.

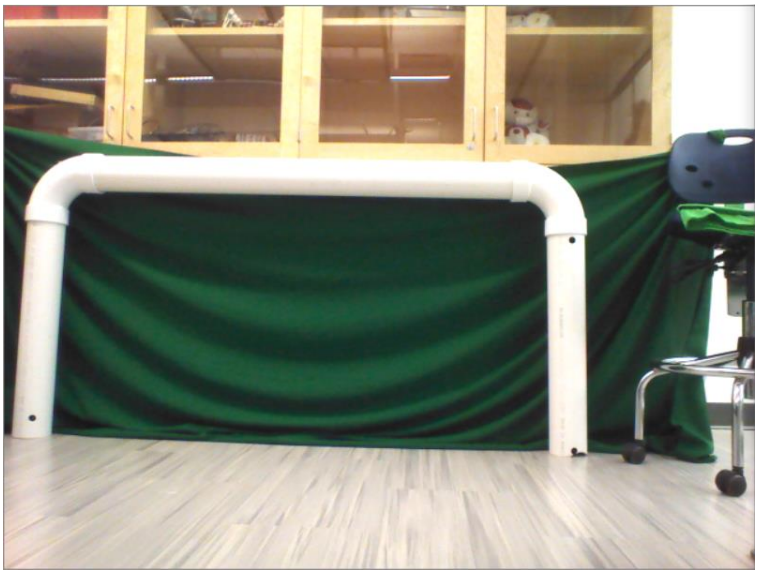

Figure 7: Unprocessed NAO View

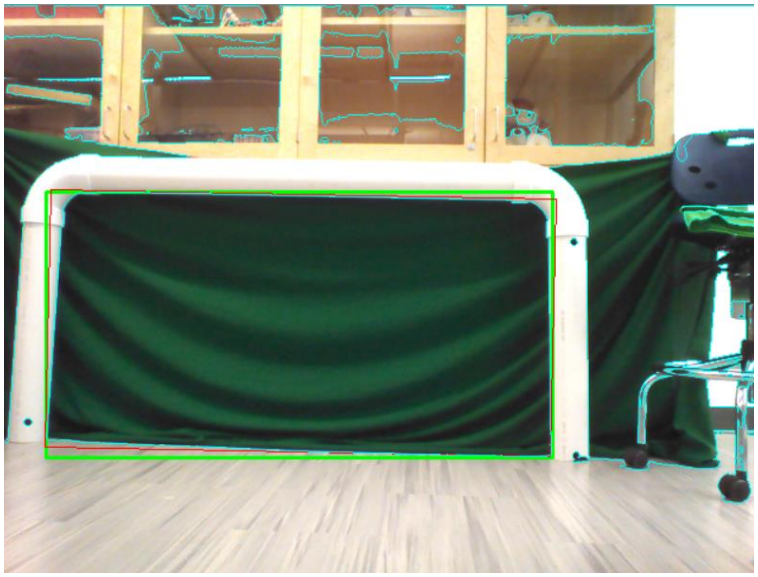

Figure 8: Goal Detected NAO View

Once the goal is detected, it must be divided into the four sections discussed in the Kicking Strategy section. This is done by dividing the goal bounding box into four equal sections, and the results are depicted in Figure 9.

Once the goal detection is complete, the NAO robot kicking the ball must also be aware of the location of the goalie in the goal. A combination of color and contour detection is used to draw a bounding box around the goalie, so it can be compared to the sections of the goal. This is shown in Figure 10.

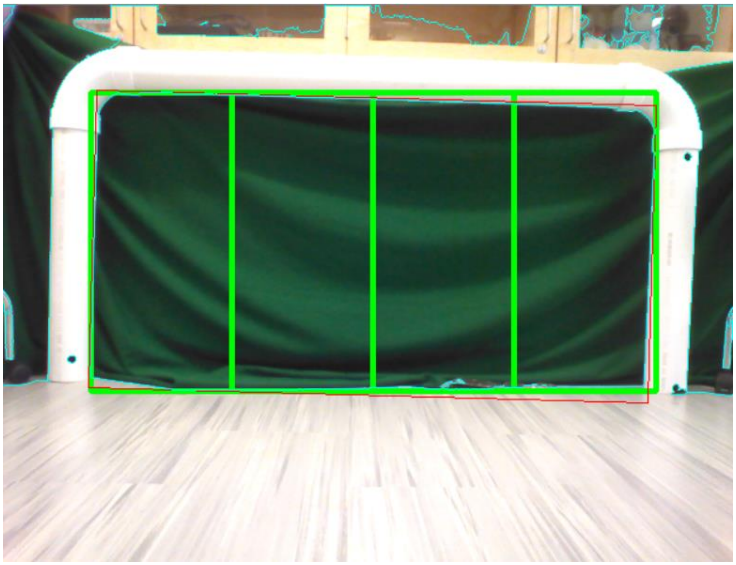

Figure 9: Goal Divided NAO View

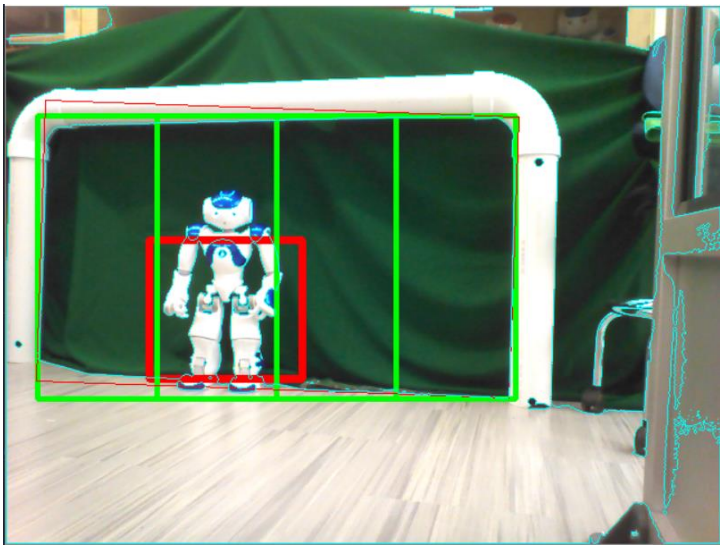

Figure 10: Goalie Detected NAO View

The goalie bounding box is compared to each of the four goal divisions, by determining which shares the most pixels. The section with the most pixels is used as the goalie's location. The results of this are shown in Figure 11. Here, section '01' is selected as the goalie's position

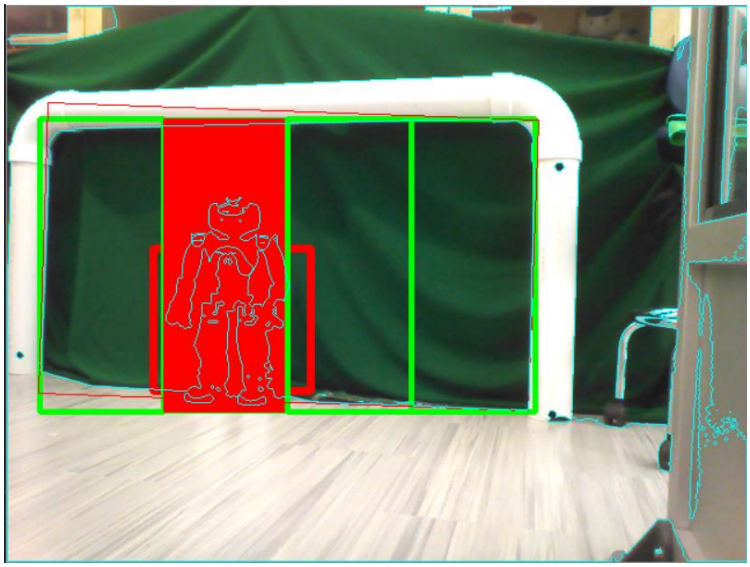

Figure 11: Goalie Section Determined NAO View

The kicking strategy algorithm is then implemented for the NAO robot to choose which section to kick towards. Based upon the kicking robot's position and the goalie robot's position, it is decided that the far-right section is the optimal area to kick towards. This is depicted in Figure 12. Because the kicking robot is closest to '10' and the goalie position is '01' section '10' is chosen for the most likely place to score a goal.

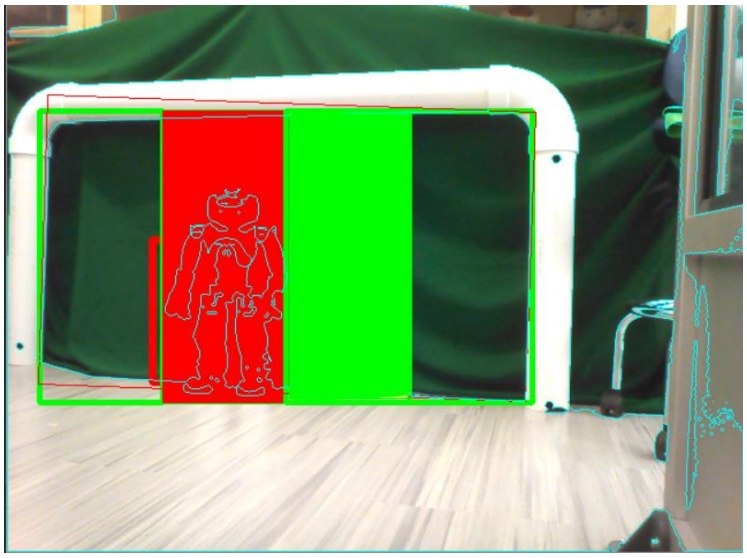

Figure 12: Shooting Section Determined NAO View

4. Results and Analysis

After running 50,000 simulations in the Yasper tool, it was found that the kicking strategy was able to score a goal 47,495 times – a 94.99% success rate. This shows the theoretical validity of the

developed kicking strategy. The developed strategy was then compared to selecting a random quadrant to kick the ball into. Out of 50,000 simulations, the random strategy was able to score a goal 38,779 times – a 77.56% success rate. This shows the validity of the produced strategy.

5. Conclusion

Enabling a NAO robot to recognize the location of the goal and the opposing team's goalie are essential aspects of developing an algorithm capable of controlling a team competing in the RoboCup. FPNs were used to develop an algorithm and test its effectiveness. The simulation results showed the advantage of deciding on a goal quadrant to kick towards versus kicking the ball randomly at the goal, regardless of the goalie's position.

6 Acknowledgement

The work was supported by the Office of Academic Affairs at The College of New Jersey, through the Mentored Undergraduate Summer Experience (MUSE) grant.

References

[1] 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," Proceedings of the IEEE International Conference on Advanced Learning Technologies, 2015, pp. 166 – 167.

[2] P. Baldoni, Y. Ynag and S. Kim, "Development of Efficient Obstacle Avoidance for a Mobile Robot using Fuzzy Petri nets," *Proceedings of the IEEE Information Reuse and Integration*, 2016, pp. 265-269.

[3] P. Cheng and K. Forward, "Fuzzy Petri nets," *Knowledge-Based Intelligent Electronic Systems, 1997. KES '97. Proceedings., 1997 First International Conference on*, Adelaide, SA, 1997, pp. 402-408 vol.2.

[4] P. Hansen, P. Franco and S-y Kim, "Soccer Ball Recognition and Distance Prediction using Fuzzy Petri nets, Proceedings of the IEEE International Conference on Information Reuse and Integration for Data Science, 2018, pp. 315-322.

[5] S-y Kim and Y. Yang, "A self-navigating robot using Fuzzy Petri nets," *Robotics and Autonomous Systems*, vol. 101, pp. 153-165

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

[7] T. Murata, "Petri nets: Properties, analysis and applications," in *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541-580, Apr 1989J.

[8] D. Ponsini, Y. Yang, and S. Kim, "Analysis of Soccer Robot Behaviors using Time Petri nets," *Proceedings of the IEEE Information Reuse and Integration*, 2016, pp. 270-274.

[9] Y. Yang, P. D. Baldoni, and S-y. Kim, "Ball Control and Position Planning Algorithms for Soccer Robots using Fuzzy Petri nets," 2016 International Conference on Computers and Their Applications, Las Vegas, Nev ada, 2016, pp. 387-392

[10] L. A. Zadeh, "Fuzzy logic = computing with words," in *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 2, pp. 103-111, May 1996

[11] Robocup Objective - *RoboCup Federation Official Website*, www.robocup.org/objective

[12] RoboCupSoccer – Standard Platform." *RoboCup Federation Official Website*, www.robocup.org/leagues/5