EPiC Series in Computing
Volume 98, 2024, Pages 116–128
Proceedings of 39th International Conference on Computers and Their Applications

Efficient Communication and Navigation for Soccer Robots using a Time Petri net Model

Shane Chiovarou, Madison Bland, and Seung-yun Kim
Electrical and Computer Engineering, The College of New Jersey, NJ, USA
Chiovas1@tcnj.edu, blandm2@tcnj.edu, and kims@tcnj.edu

Abstract

Robotics is a very expansive and ever-growing field that allows for a large number of different scenarios to occur. Once application is coordination of multiple robots in a single system, and that is highlighted in an environment such as RoboCup competitions. The Contextual Communication and Navigation (CCN) Algorithm is an approach to the problem of multiple robots coordinating at a common goal through the lens of Communication and Navigation. A scheme of communication that allows Nao Robots to efficiently communicate their positions using the minimal number of bits to transfer data was designed as well as a navigation scheme that allows robots to use the information they have on other robots to navigate to a goal. The CCN Algorithm was simulated and tested along with Dijkstra and A* in order to compare the effectiveness of each of the algorithms in tackling this problem and it was found that the CCN Algorithm was the most efficient.

1 Introduction

Robots are being used more and more to accomplish many different tasks that are only increasing in complexity. Some tasks require multiple robots working in conjunction to accomplish some goal, and in a case like this, the robots involved need to be constantly communicating with each other in order to be efficient in their tasks. This communication is much faster and more efficient than people communicating. There is no need for audio or visual signals to be passed due to electrical signals efficiently carrying the data from robot to robot. The data is more efficient than people communicating due to the lack of emotion/inflection, the robots receive the data as data with no other influence. One method of electronic communication is email, which has been utilized in robots in many different ways. For example, a scheme utilized email when reading business cards to send a message to the card owner (Zhao, 2018) and a method employed email to send voicemail through the RAPP system (Figat, 2015). Email is also very prevalent in security robot applications, Researchers utilized email to notify proper personnel of potential security threats (Bedi, 2010, Bhagwat, 2015, Gahlaut, 2017).

Another side of communication involving robots is human-robot interaction. This is a large area that strives to make interactions with robots as comfortable and natural as possible for people. Tan et al.

determined that the way robots in a system interact with each other impacts how a person will view the robots (Tan, 2019). In order to design and verify the complex algorithms that need to be developed for these systems, Petri net (PN) modeling is used. Petri nets are modeling diagrams conceptually similar (but much more operationally complicated) to flow charts, allowing for easy observation of the sequence of events that are taking place. Petri nets have been used to model many different kinds of situations, for example, the modeling of cooperation and communication between robots (Weissman, 2018), an emergency healthcare system (Zhou, 2019), and Cyber-Physical Systems (Yang, 2018).

This paper consists of the following sections: Section 2 provides background information on Petri nets, NAO robots, RoboCup, and email communication. Section 3 breaks down the communication and navigation algorithms. Section 4 discusses the results. Finally, Section 5 concludes the paper by wrapping up everything and discussing potential future avenues of work.

2 Background

2.1 Petri nets

A Petri net (PN) is a graphical and mathematical modeling tool that allows for the visualization and simulation of a variety of simple and complex systems. Petri nets are defined as:

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

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 M_0 is the initial marking. In graphical representations of Petri nets, places are denoted with circles, transitions with rectangles, and arcs with arrows. Places in the net can be considered states of the system that is being modeled, and transitions can be considered the operations that must occur to switch from one state to another. Tokens, which are the solid circles inside of places, allow the state of the system to be visually observed in real-time (Murata, 2013).

Additional logic and conditions can be incorporated into a Petri net using different types of arcs, specifically inhibitor and read arcs. These two arc types are effective opposites with the inhibitor arc disabling a transition when active and the read arc enabling a transition when active. They appear like normal arcs, but the arrow terminates with a circle instead of an arrowhead. The Petri net seen in Figure 1 was created using the program HPetriSim[*].

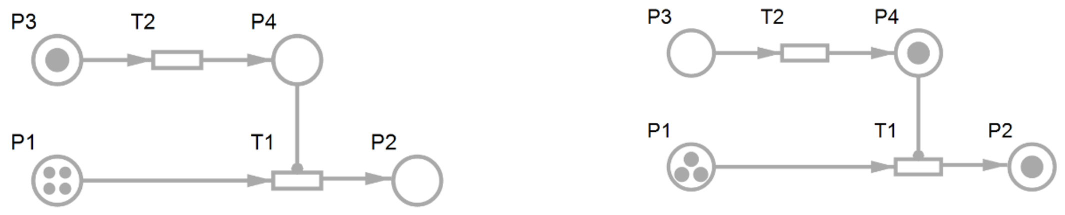


Figure 1: Petri net using Inhibitor Arc in HPetriSim

Other programs, such as TINA, are more useful for more complex types of Petri nets. However, it is important to note some visual differences between the two programs. The main difference is the depiction of inhibitor arcs, in HPetriSim these arcs are shaded circles but in TINA the circles are not shaded. Read arcs do not exist in HPetriSim, and TINA uses a shaded circle, shown in Figure 2. In the

[*] https://github.com/Uzuul23/HPetriSim

case shown in Figure 1, when place P4 is empty the transition T1 is able to fire a token from P1 to P2. However, once T2 fires the token from P3 to P4, the inhibitor arc going from P4 to T1 is activated and T1 can no longer fire tokens from P1 to P2. Additional logic also comes in the form or arc weights. These weights are associated with each individual arc and represent the minimum number of tokens necessary in order to fire the transition.

2.2 Time/Timed Petri nets

Although PNs themselves are a very powerful tool, there are limits to the models that can be developed with the base version. In order to add more complexity and robustness, different variations of Petri nets have been developed. Fuzzy logic was added to PNs in order to create Fuzzy Petri nets (FPNs) and shown to be effective in modeling robot navigations (Kim, 2016). Time intervals were incorporated to transitions in PNs to developed Time Petri nets (TPNs) and used delaying token firings (Ponsini, 2015), and colored tokens were used to form Colored Petri nets (CPNs) and modeled different types of data with different colors (Pham, 2015). One such variant is the Timed Petri net (TdPN) but a Timed Petri net is a subset of TPN. It is important to first explain what a TPN is before diving deeper into TdPN.

Time Petri nets introduce time and durations into a Petri net, in this specific case we will focus on time elements being added to Transitions alone. For a TPN, each transition has two times associated with it $[a_i, b_i]$ with $a_i \leq b_i$. a_i is the time at which the transition is able to fire, and b_i is the time at which the transition is guaranteed to fire.

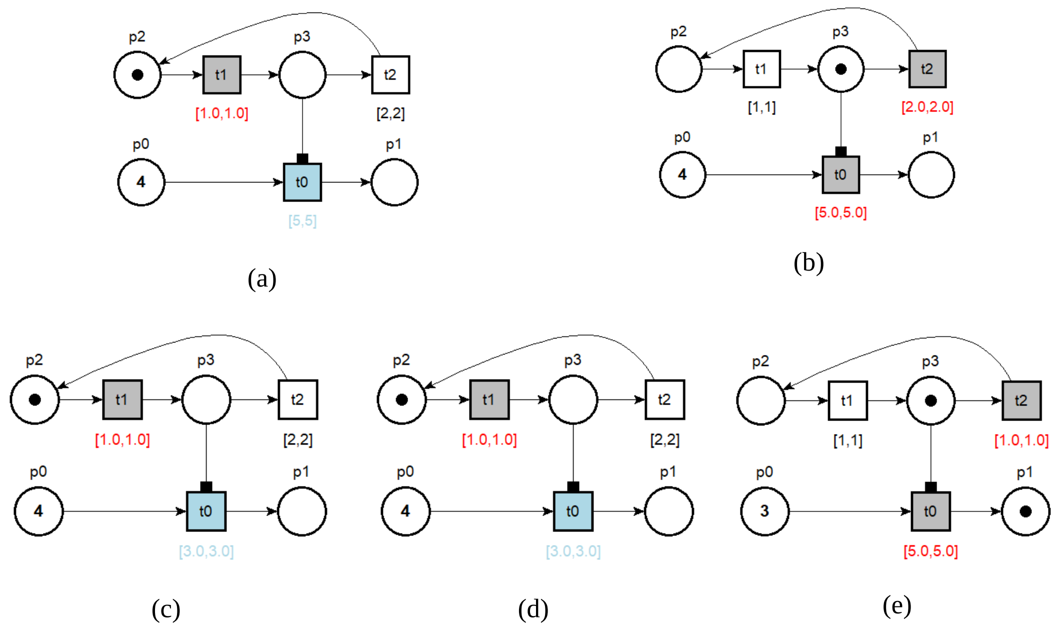


Figure 2: Timed Petri net with Stopwatch Arc in TINA

A Timed Petri net (TdPN) is a specific case of the TPN where $a_i = b_i$. This effectively gives each transition only one time value, and this can be thought of as the duration it takes for that transition to fire. Timed Petri nets are formally defined as the following:

$$\text{TdPN} = (P, T, A, W, M_0, \tau)$$

Where each definition is the same as a regular PN. The only difference is the addition of τ, which represents the time delay associated with each transition in the TdPN. The delay begins once the transition is able to fire, then a timer starts and once the timer ends the transition is able to fire (Popova, 2013). Despite only incorporating time into transitions, new arcs are introduced into a TdPN that can affect the flow of time in Transitions. The two new arcs are Stopwatch Inhibitor and Stopwatch. These arcs are very similar to the Inhibitor and Read arcs of a regular PN, but they affect the timer of the transition rather than the transition itself. The Stopwatch Inhibitor disables the timer on a Transition once it is activated, but it does not reset the time like a regular inhibitor would. The Stopwatch is functionally the opposite of the Stopwatch Inhibitor, it enables the timer when activated and retains the time left when deactivated. These arcs exist only in TINA and they are represented with a line terminating with an open square for Stopwatch Inhibitors and a line terminating with a shaded square for Stopwatch arcs. An example of the Stopwatch arc in action can be seen Figure 2.

In Figure 2 the Stopwatch arc connects place p3 and transition t0. In part (a) the 5 seconds timer on t0 should be running, but since there is no token in p3 the stopwatch arc is disabled which prevents t0 from running. In (b) once t1 is fired p3 has a token and t0 begins to count down. In parts (c) and (d) t0 is interrupted once 2 seconds pass and t2 fires the token from p3 back to p2, the stopwatch arc is once again disabled but the 2 seconds that had passed was retained in t0, which now only has 3 seconds left. Part (e) shows the result of the third cycle of the top token, 5 seconds have passed for t0, so it now fires and resets, leaving 1 second on t2. This process will repeat for each of the 4 tokens in p0, and when all 4 are in p1 the top token will continue cycling through p2 and p3 forever. Figure 2 was created in the program **TI**me Petri **N**et **A**nalyzer (**TINA**)[†]. Included with TINA is a variety of tools that allow for graphical or textual construction of PNs TPNs, analysis of the structure of the net, or a stepper simulator. The NetDraw (nd) tool was utilized for the creation and graphical representation of TPNs, as well as the simulation of those nets.

2.3 Nao Humanoid Robots and RoboCup Competition

The Nao Humanoid robot is designed by Aldebaran, which emulates the movement and behavior of humans utilizing various motors and sensors[‡]. The motors located all throughout the body allow for 25 degrees of freedom and the sensors include two cameras as well as multiple touch sensors throughout the body. The Nao robot has been utilized in many human-based applications, such as human emulation and interaction. These applications include movement emulation, applying some kind of artificial intelligence to the robot so it accomplishes a task. Other applications involve assistance for certain groups of people that require it or may benefit, such as elderly or children with Autism Spectrum Disorder.

The Nao robot stands 57.4 cm tall, weighs 5.4 kg, and is powered by a 1.6 GHz processor. These specifications, most importantly the processor speed, must always be considered when approaching any problem. For example, the processor speed puts a limit on the efficiency of the programs or algorithms implemented into the robot, if the process is too resource intensive the task will not be carried out efficiently or reasonably. Provided with the NAO Robot is a programming environment called Choregraphe. This environment allows for different modules to be combined to form a stream of execution, it also allows for custom Python code to be written and run on the robots.

RoboCup competition is a yearly event that aims to promote robotics and AI research. The long-term goal is to create "a team of fully autonomous humanoid robot soccer players that shall win a soccer game against the winner of the most recent World Cup." There are different leagues that facilitate different aspects of this goal. The one of interest is the Standard Platform League (SPL), in which all

[†] https://projects.laas.fr/tina/download.php

[‡] http://doc.aldebaran.com/2-1/nao/index.html

teams use the same Humanoid Robot, in this case, it will be the Nao Robot. The SPL matches in RoboCup take place in a field that is six meters wide by nine meters long. Much like a soccer field which this is emulating, the field has two goals on either side of the field with traditional soccer field markings. Unlike soccer, there are only five players per team, including the goalie.

2.4 Email Communication between Nao Robots

In the field of robotics, the main purpose of email is usually the last step which involves sending some data to a person. The common practice is the robot performs whatever task it was designed to do, then once something noteworthy occurs or something goes wrong. This can be seen in medical and security applications. In the instance of security as accomplished by, there is a network of cameras and sensors watching over an area. Once movement is detected an image is taken, or in some cases a video, and it is emailed to the correct personnel. One thing that has been very scarcely investigated is the potential of robots passing data to each other using email. This means of communication almost always involves a human as either one of the parties communicating or as an intermediate step in the transfer of data.

Nao robots have the feature to send emails to a preselected account, this can be a basic text message, or it can include attachments such as images. This function is fairly common in robotics, such as in the email applications previously mentioned, these robots are sending emails to people. However, Nao robots also have the capability to fetch emails from an account. This built-in function raised the question of what if one robot sent an email with a piece of data, another robot receives that email and processes the data to make a decision.

Email communication was used in this research project to serve as a proof of concept for the algorithms at work. This will allow for easy logging and troubleshooting with the communication between agents in the system. Communication standards and systems already exist, and future implementation will use them, but for this proof of concept will utilize email to verify the other algorithms at work.

3 Contextual Communication and Navigation Algorithm

A system that would allow the robots on the same team to efficiently coordinate movements and actions in the RoboCup SPL was developed into the Contextual Communication and Navigation (CCN) Algorithm. The proposed algorithm, which includes Communication Module (CM) and Navigation Module (NM), demonstrates how the different robots on the field would communicate with each other as well as what to do with the information they receive, which in this case is a context-aware decision on where to navigate on the field.

3.1 Communication Module

The communication between the robots on the field must be consistent under a single, designed standard that defines each type of message that could be passed through the system. In this case, the communication will always be between a player and the controller. Messages that the player needs to be able to send its position to the controller and receive anything from the controller. The controller needs to be able to receive player positions and broadcast either player positions or the ball position. In order to minimize the size of the message, the minimum number of necessary bits will be used. There will also be a special character used in the messages, which is the semicolon. A single semicolon denotes the end of a piece of the message, and two semicolons denotes the end of the broadcasted message. The format of all messages as well as how many bits are required for that piece of information can be seen Table 1.

Message Portion	Player/Ball	ID	Top/Bottom	x	y
Number of Bits	1	2	1	4	4

Table 1: Message Format

The first bit is used to differentiate between the two different kinds of information that could be broadcast using these messages, which is either a player position or the ball position. The ID is used to identify what player the positional data is referring to since the positional data must remain associated with the correct player. There are only four unique IDs that can be made with two bits, but since the goalie robot will remain in the goal there is no need for it to be assigned an ID. The next three pieces of data, and nine bits, breaks down the exact location of the robot on the field. This will be discussed further in the Navigation Module of the CCN Algorithm.

The Communication Module of the CCN Algorithm is best portrayed as a series of functions that have their own separate thread of execution as shown in Figure 3 through Figure 7.

```
WAIT (ID * 0.7)
WHILE (true) DO
    oldPos = currentPos
    WHILE (oldPos == currentPos) DO
        WAIT (2.8)
    END
    body =
encodeMessage(0,ID,currentPos) + “;”
    sendEmail(“”,body,controllerEmail)
END
```

Figure 3: Player sends its position to the Controller

```
WHILE (true) DO
    WHILE (!emailReceived) DO
        WAIT (1.4)
    END
    body = readEmail(0)[1]
    message = decodeMessage(body)
    FOR (i = 0; I <= message.length; i++)
        IF (message[i][0]==0)
            updatePos(message[i])
        ELSE
            nav(message[i])
    END
END
```

Figure 4: Player receives any message from the Controller

```
WHILE (true) DO
    WHILE (!emailReceived) DO
        WAIT (0.7)
    END
    body = readEmail(0)[1]
    message = decodeMessage(body)
    updatePos(message)
END
```

Figure 5: Controller Receives a position from any Player

```
WHILE (true) DO
    WAIT (2.8)
    body =
encodeMessage(0,00,getPos(0)) +
encodeMessage(0,01,getPos(1)) +
encodeMessage(0,10,getPos(2)) +
encodeMessage(0,11,getPos(3)) + “;”
    FOR(i = 0; i < 4; i++)
        sendEmail(“”,body,playerEmail[i])
    END
END
```

Figure 6: Controller broadcasts all Player positions

```
WHILE (true) DO
    oldBPos = currentBPos
    WHILE (oldBPos == currentBPos) DO
        WAIT (2.8)
    END
    body = encodeMessage(1,00,currentBPos) + ";"
    FOR(i = 0; i < 4; i++)
        sendEmail("",body,playerEmail[i])
    END
END
```

Figure 7: Controller broadcasts the Ball position

3.2 Navigation Module

The Navigation Module (NM) of the CCN algorithm aims to abstract some of the nuances and precision from robot movement away from what needs to be communicated. This is accomplished by partitioning the field into sectors. These sectors would be equal size and evenly distributed throughout the entire field, and they will be used to approximate the position of the robots. The size of the sectors is extremely important for the overall effectiveness of the CCN Algorithm, since only one robot will be able to inhabit a sector at a time, therefore the physical attributes of the NAO Robot needed to be considered.

The length of each sector was determined using the size of the NAO Robot's foot. The NAO foot is 150 mm in length. It was decided that the size of the sector would be double this amount which makes the sector 300 mm in depth. This amount was decided since it allows two NAO bots to stand in line in the center of two subsequent sectors without interference.

The width of the sector was determined using the typical movements and positions of the NAO Robot. Typical implementations in RoboCup have the NAO robot with its arms crossed behind its back, with its elbows bowed outward. The width of the robot while in this position is 300 mm. This would be the width of the sector however the NAO Robot sways when walking, and this displacement measures around 50 mm on each size, so this amount is added to the width for a total sector width of 400 mm, as shown in Figure 8.

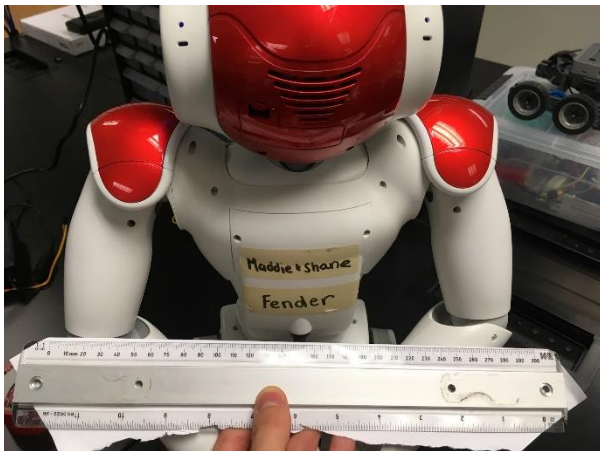


Figure 8: NAO Robot Width Measurement

Each sector 400 mm wide by 300 mm long which partitions the field into a 30 by 15 grid. The CM of the CCN Algorithm uses four bits for the x and y coordinates on this grid, and the top/bottom bit indicates which half of the field the robot is on. Testing was also done to determine the length of time it takes the Nao Robot to move between sectors. Many measurements must be taken since the sectors are rectangular. Multiple trials were taken to find that the average times for traversal up and down the field, across the field, and diagonally to take 4.5, 3.5, and 5.6 seconds respectively.

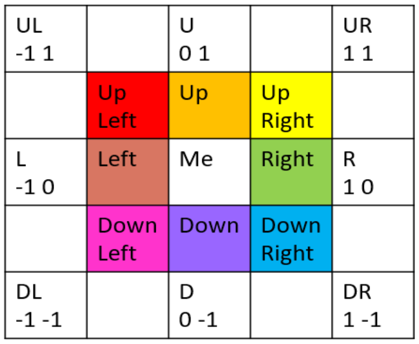
The NM of the CCN Algorithm operates on the principle that we know the context within which the robot will need to navigate. The field is the same size every time and it will be split up the same way every time. The task for any robot will be to reach a goal position, which will be the location of the soccer ball. Navigation to that point is done using a simple assessment of the current state of the field and moving to a new sector based on that state. A new destination will be calculated each time the robot is at a new sector, which will be referred to as a step. There are eight directions to choose from for each step, and a new sector goal will be decided with a period based on the longest step travel time, which is diagonal as shown in Figure 9.

```
SWITCH (diff = goalI – playerI) //Assign Horizontal
    CASE (diff < 0)
        dir[0] = -1
    CASE (diff > 0)
        dir[0] = 1
    CASE (diff == 0)
        dir[0] = 0
END

SWITCH (diff = goalJ – playerJ) //Assign Vertical
    CASE (diff < 0)
        dir[1] = -1
    CASE (diff > 0)
        dir[1] = 1
    CASE (diff == 0)
        dir[1] = 0
END

IF (adj[dir[0] + 1, dir[1] + 1]) // Obstruction
    IF (dir[0] != 0 && dir[1] != 0) // Diagonal
        IF (abs(goalI – playerI) >= abs(goalJ – playerJ) //move to reduce total distance to goal
            dir[1] = 0
        ELSE
            dir[0] = 0
    ELSE IF (dir[1] != 0) //Up or Down
        IF (abs(playerI) >= 8) //Move toward center
            dir[0] = -1
        ELSE
            dir[0] = 1
    ELSE //Left or Right
        dir[1] = -1 //down to be defensive
```

Figure 9: Navigation Module of the CCN Algorithm

The algorithm categorizes movement as a two-dimensional vector, with a vertical and horizontal component. In the horizontal direction, negative one means left, one means right, and zero means neither, with the vertical direction similarly corresponding to down and up respectively. The input into the algorithm is the current position of the player, the current position of the goal, and a matrix representing the sectors immediately adjacent to the player. The directional vector is the output of the algorithm, and the robot will move according to it. The construction of this vector is split into three main parts, the horizontal, the vertical, and the collision check. The horizontal and vertical components are assigned based on the difference between the coordinates of the player and the ball. If the difference is negative, a negative one is assigned, and if the difference is positive, a positive one is assigned. Therefore, if the directional vector is currently (1,1), then the CCN Algorithm plans to have the robot move up and to the right, as shown Figure 10.

The directional vector at this point represents the desired direction of travel for the player to get as close to the goal as possible. However, since there is the possibility of being blocked, a collision check must be done to adjust the planned motion, if necessary. This is done by checking the adjacent sector matrix, if the desired sector is occupied, a new destination sector must be assigned. The type of reassignment depends on the properties of the desired movement. If the desired move was diagonal, then the robot must move in a cardinal direction, with the one that will decrease the absolute distance to the goal the most being decided. This concept is visualized in Figure 11.



UL -1 1		U 0 1		UR 1 1
	Up Left	Up	Up Right	
L -1 0	Left	Me	Right	R 1 0
	Down Left	Down	Down Right	
DL -1 -1		D 0 -1		DR 1 -1

Figure 10: Defining Directions

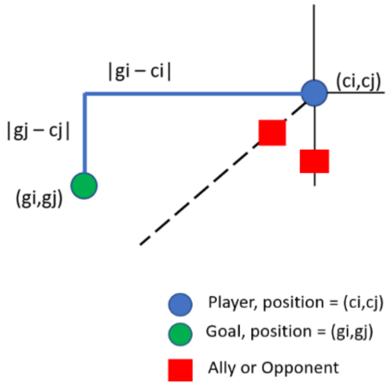


Figure 11: Deciding Directions

When the desired movement is in a cardinal direction, a diagonal movement will be substituted. The exact direction of this diagonal movement is determined by general strategy in a game of soccer. When a vertical movement is blocked, the player will move diagonally toward the center of the field, which allows for more options on the next movement calculation. When a horizontal movement is blocked, the player will default to move diagonally downward in the desired horizontal direction, this is a defensive play against the opponent blocking.

4 Results

The CCN Algorithm will be judged under two different metrics to determine its effectiveness. The first evaluates the Communication portion of the CCN Algorithm, and this is simply its ability to be implemented effectively. The Navigation portion of the CCN Algorithm will be compared to other path planning algorithms, which are Dijkstra and A* in this case.

4.1 Choregraphe Communication

Two models were created in Choregraphe that serve as block diagrams that demonstrate the flow of execution. Each of the models can be split into different portions that accomplish each of the desired tasks the CCN Algorithm describes. The controller and player have different models which demonstrate their high-level structure, and they can be seen in Figure 12.

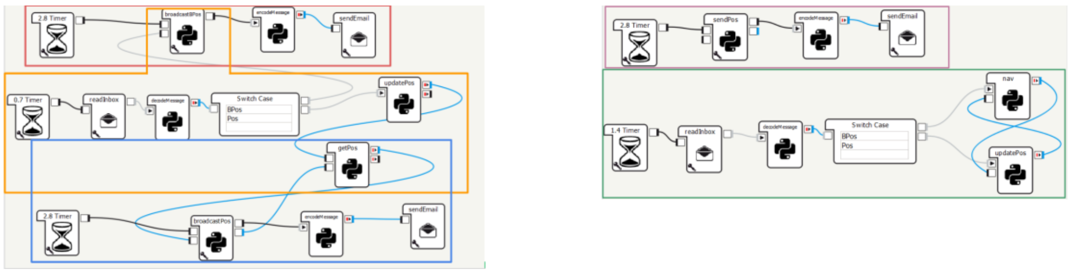
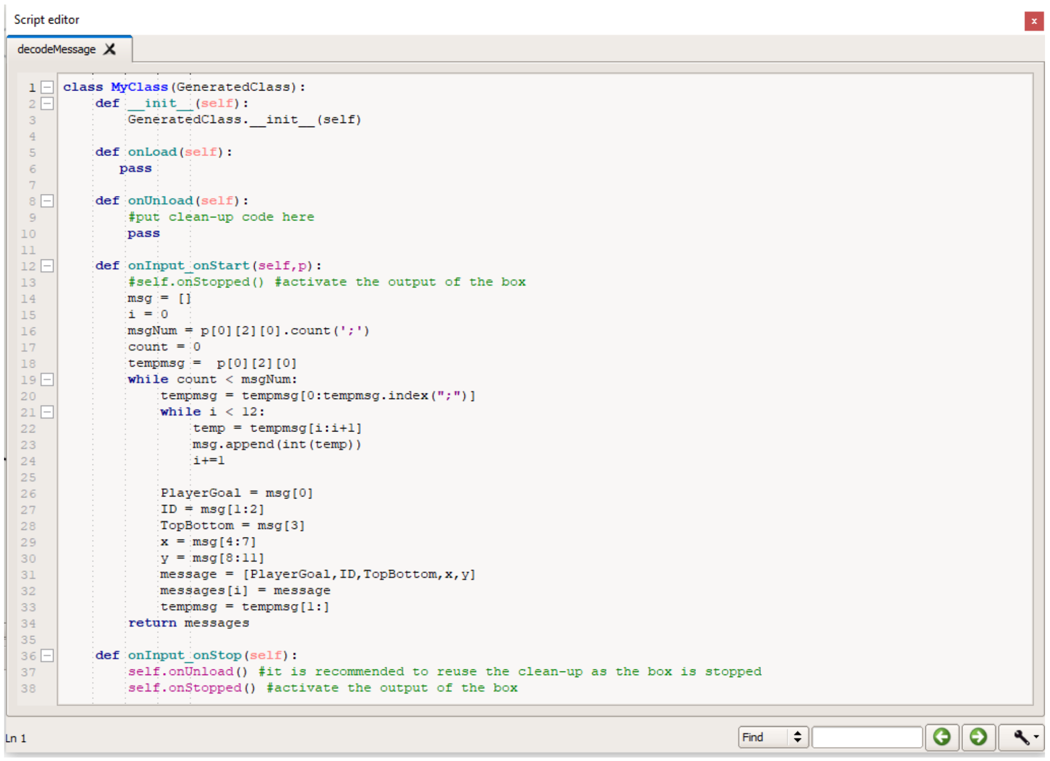


Figure12: Choregraphe Models

An implementation of the Decode Block was done to demonstrate the ease of following the algorithm to lead to a full implementation, as shown in Figure 13. It is also important to mention that the algorithm runs in negligible time compared to the multiple seconds it takes for a robot to move between two sectors. Therefore, the synchronization and timing is not an issue in the implementation of the CCN Algorithm.



```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
       pass

    def onUnload(self):
        #put clean-up code here
        pass

    def onInput_onStart(self,p):
        #self.onStopped() #activate the output of the box
        msg = []
        i = 0
        msgNum = p[0][2][0].count(';')
        count = 0
        tempmsg =  p[0][2][0]
        while count < msgNum:
            tempmsg = tempmsg[0:tempmsg.index(";")]
            while i < 12:
                temp = tempmsg[i:i+1]
                msg.append(int(temp))
                i+=1

            PlayerGoal = msg[0]
            ID = msg[1:2]
            TopBottom = msg[3]
            x = msg[4:7]
            y = msg[8:11]
            message = [PlayerGoal,ID,TopBottom,x,y]
            messages[i] = message
            tempmsg = tempmsg[1:]
        return messages

    def onInput_onStop(self):
        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
        self.onStopped() #activate the output of the box
```

Figure 13: Choregraphe Python Implementation of Decode Block

4.2 Navigation TPN Simulation

In order to evaluate the effectiveness of the CCN Algorithm when compared to others that could be used for navigation, a model must be created. A TPN that models the RoboCup field was created. The places represent sectors and transitions represents a move between two sectors. It is also possible to set the goal position by altering the grid of places on the bottom right of the model as well as set the positions of other players on the field.

The TPN is a scale model of the full field since only a portion is necessary to evaluate the effectiveness as well as the fact that the complexity of creating the model increases quadratically as the size of the grid grows. In this case, a five-by-five grid of sectors was made and is sufficient for testing. The times associated with each of the transitions is what was measured for the time it takes the robot to traverse between sectors in that direction. The TPN in its entirety can be seen in Figure 14.

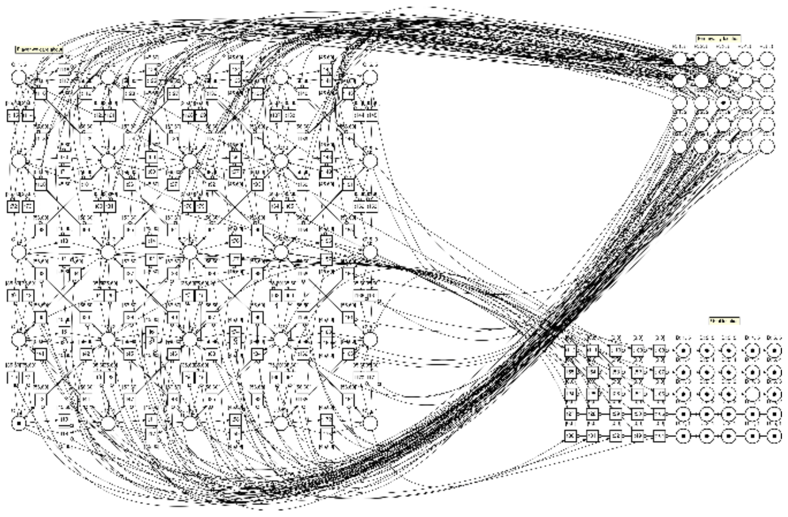


Figure 14: TPN Model of the CCN Algorithm

Four different algorithms will be compared to each other using the TPN, with their efficiency being used as the point of comparison. The four algorithms are: Random, Dijkstra, A*, and CCN. Efficiency must be defined to give the algorithms a metric for comparison, and it will be defined as follows:

$$Efficiency = 2 * \frac{Actual\ Node\ Visits}{Total\ Node\ Visits}$$

An actual node visit is defined as the number of nodes that the robot would physically visit when navigating from a start position to a goal position. A virtual node visit is defined as the nodes that needed to be checked when determining the path for the robot. The total node visits is defined as the sum of the actual node visits and the virtual node visits. The quotient is multiplied by two in order to normalize the efficiency rating to one since a perfect navigation will check the same number as it visits which makes the total double the actual. Efficiency is measured on a scale of zero to one, with the higher efficiency being better.

Multiple trials were conducted to get a comprehensive comparison of the different algorithms in different situations. These trials placed the goal position and other player in different positions and ran the algorithms, with the virtual and actual node visits being recorded.

An example trial has the start position in the bottom left of the grid, with another player two sectors away diagonally up and to the right. The goal is one sector to the right of the other player. The results of this trial are shown in Table 2.

Trial	Virtual Node Visits	Actual Node Visits	Total Node Visits	Efficiency
Random	504	63	567	0.1111
Dijkstra	29	3	32	0.1875
A*	15	3	18	0.3333
CCN	4	3	7	0.8571

Table 2: Results of Trials

This process was repeated over 20 times to cover a variety of situations in order to fully assess each of the algorithms in comparison to each other. The results for efficiency were compiled into an average efficiency of the algorithm which can be used for comparison. The average efficiency values for all four algorithms can be seen in Figure 15.

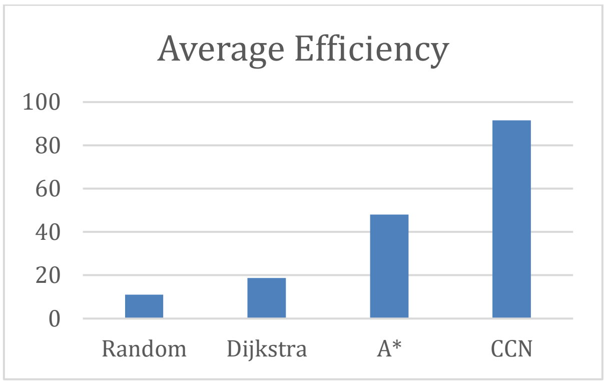


Figure 15: Efficiency of Algorithms

The CCN Algorithm is able to perform much more efficiently when compared to the other algorithms. Although other two navigation algorithms, Dijkstra and A*, both perform better than random movement but the CCN Algorithm has the highest average efficiency of 0.9153.

5 Conclusions

In recent years, robots have become an ever-increasing presence in the world. This increase in robots has also prompted the need for coordination and communication between all of the robots. The Contextual Communication and Navigation Algorithm was designed to take a specific context, RoboCup SPL, and implement an algorithm that would efficiency handle the situation. This scenario was modeled using a Time Petri net, with durations for events carefully selected and allocated based on measurements and testing. Once the CCN Algorithm was fully developed, it was compared to different algorithms that accomplish the same task. It was found that the CCN Algorithm was able to accomplish this task of navigating the robot to the goal position was more efficient than the other algorithms.

Future work on this algorithm may include further implementation of the algorithm in Python, which would allow the system to be run outside of simulation on a team of Nao Robots. Due to the modular design of the CCN algorithm, it can be applied to different scenarios where multiple robots need to communicate with each other.

References

P. Bedi, R. Singh and T. K. Matharu (2010), "Ensuring security in a closed region using robot," *2010 IEEE Intnl Conf. on Computational Intelligence and Computing Research*, , 2010, pp. 1-4.

C. N. Bhagwat (2015), Y. N. Krishnan and K. Badrinath, "Cloud based intruder detection system," *International Conference on Electronics and Communication Systems (ICECS)*, Coimbatore, 2015, pp. 1244-1246.

M. Figat, T. Kornuta, M. Szlenk and C. Zieliński (2015), "Distributed, reconfigurable architecture for robot companions exemplified by a voice-mail application," *International Conference on Methods and Models in Automation and Robotics (MMAR)*, Miedzyzdroje, 2015, pp. 1092-1097.

S. Gahlaut and K. R. Seeja (2017), "IoT based smart campus," *International Conf. on Innovations in Control, Communication and Information Systems (ICICCI)*, Greater Noida, India, 2017, pp. 1-4.

S. Kim and Yilin Yang (2018), “A Self-Navigating robot using Fuzzy Petri nets”, Robotics and Autonomous Systems, Vol. 101, 2018, pp. 153 – 165.

T. Murata (2013), “Petri Nets: Properties, Analysis and Applications,” *Proc. of the IEEE*, Vol. 77, 2013, pp. 541-574

K. Pham, C. Cantone and S. Kim (2017), “Colored Petri net Representation of Logical and Decisive Passing Algorithm for Humanoid Soccer Robots”, Proceedings of the IEEE Information Reuse and Integration, 2017, pp. 263 – 269.

D. Ponsini, Y. Yand, and S. Kim (2015), “Analysis of Soccer Robot Behaviors using Time Petri nets, Proceedings of the IEEE Information Reuse and Integration, 2015, pp. 270 – 274.

L. Popova-Zeugmann (2013), Time and Petri nets. Springer.

X. Z. Tan, S. Reig, E. J. Carter and A. Steinfeld (2019), "From One to Another: How Robot-Robot Interaction Affects Users' Perceptions Following a Transition Between Robots," *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 114-122.

M. Weissman, D. Ponsini and S. Kim (2018), "Prioritized Situation Awareness for Soccer Robots Using Timed Transition Petri Nets," *IEEE International Conference on Information Reuse and Integration*, pp. 134-138.

Y. Yang, W. Xu, S. Wang and K. Wei (2018), "Modeling and Analysis of CPS Availability Based on the Object-oriented Timed Petri Nets," *Chinese Control Conference (CCC)*, pp. 6172-6177.

X. Zhao, Q. Gao, S. Shen and Z. Wang (2018), "Business Card Recognition and E-mail Delivery Based on NAO Robot," *Chinese Control And Decision Conference (CCDC)*, pp. 5595-5599.

J. Zhou, J. Wang and J. Wang (2019), "A simulation engine for stochastic timed petri nets and application to emergency healthcare systems," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 969-9809