Controller for Unity 3D based on Rob Soccer simulation

Jinghong Xiao
School of Computing
Queen's University
Kingston, ON, Canada
16jx12@queensu.ca

Xuebin Yang
School of Computing
Queen's University
Kingston, ON, Canada
15xby@queensu.ca

***Abstract*—Robot Soccer is a simulation game that incorporates the challenging robot control techniques aiming to provide automatic control of one participant in the soccer game applying the use of game strategy. It also includes a number of challenges in the areas of artificial intelligence and image analysis. In this paper, we would like to discuss different designs and implementation methods for creating automatic robots in the Unity 3D soccer simulation environment with a TCP connection from controller to Unity. Moreover, we propose a basic automatic controller based on geometry theory and provide details about our approach to improve robots' skills and localization.**

***Keywords*—Unity, TCP connection, geometry theory, 3D Soccer Simulation**

Introduction

Robot soccer refers to soccer games played by software agents instead of real robots. Although enabling robots play soccer does not yield any significant impact on society, it would be a great achievement in the fields of AI and robotics. In addition, the knowledge gained by developers in the process would possibly become useful in a more productive way in the future. It covers many challenging fields of robot control and includes the game strategy, which is accessible and valuable for future researches. Robot soccer simulation has played an essential part of robot soccer. The robot soccer simulation is built based on Unity 3D, which is a powerful cross-platform 3D engine and a user-friendly development environment for 3D games. 3D simulation is widely applied to increase the realism of the simulated environment with extra dimension and more complex physics [1].

The motivation of the project is to bring out the way that the red player performs smartly to track the ball and play against the blue player in the ready-made simulator. Our objective is to let the red player win the game with a higher score. To win the game, the design of the controller is supposed to direct the actions of the red player constantly according to the dynamic game situations with real-time information [2]. The interesting thing about the project is that it contains many challenges which we could apply our learned knowledge to solve the simulation. The process of solving problems is passionate and satisfying. Moreover, RoboCup obtaining increases popularity these days, from its original mission:

"By mid 21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup." [3]

Robots are not as smart as humans, so this nature of non-human thinking mechanism draws our attention to developing the robot soccer system.

The main goal of this paper is to implement a simple game strategy for Robo Soccer 3D Simulation, focusing on implementing TCP connection from controller to Unity and creating an automatic control model with the strategy algorithms of the red player. The situation is read and analyzed based on the postures of the robots and the ball position. It will explore the problems and qualities of these strategies when the red player plays against the blue player. Due to the limited time available for this project, unfortunately no machine learning will be used, and the strategies implemented here will only use geometry theory methods. This will be further discussed in the description section. Many problems arise when trying to create the best simulation. Getting the rules to take into account all the possible scenarios in the game is time-consuming, if not impossible. However, a reasonably good algorithm work is possible, and even if it is far from optimal, it will display an artificially intelligent agent that works automatically.

Background

The robot soccer simulation is a simplified RoboCuo-type game created by using Unity 3D. The simulation contains a blue player, a ball and a red player [4]. The blue player is controlled by the keyboard whereas the red player is controlled by the model which we are going to design and develop.

Fig. 1. Unity 3D environment

a) Rule of the simulation game: As the Figure-1 shows, at the beginning of the simulation, the ball lies on the standard kick off position in the central position of the 3D environment and both of the blue player and the red player are on their respective side with equal distance away from the ball. The simulation lasts a predetermined time interval of 240 seconds. Each position has its coordinates, according to the position of the player and the position of the robot to achieve the correct catch. Upon acquiring the ball, the player has a fixed time period to possess the ball. If the possession time period expires, the player would eject the ball and suspend for a fixed time period and the player is not allowed to move for a certain amount of time. The simulation environment will reset when a player gets the goal.

b) Commands: The simulation contains many supported commands which are ready to use. From the Table-1, moveForward() allows the robot to move forwards or backwards corresponding to the given positive integer or negative integer and number 0 means stop moving. Both Spin() and moveRight() functions have the same property as moveForward(). GPS() allows the robot to obtain the coordinates of itself, the blue player and the ball. GetCompass() allows the robot to determine the compass that it is facing. setSuction() triggers the robot to suck the ball with the power specified by a positive integer and shoot the ball by a negative integer. To develop the best controller of the red player, our first step is familiar with these commands and know when these commands should be used to produce the best result. Hence, an effective idea of controller design is needed. Moreover, A deep understanding in how TCP works is also required together with a strong mathematical background to best support the implementation of the controller.

Function name	Description
moveForward(<Real>)	Move the object forwards at the speed specified by the parameter (-100 to 100), a negative parameter value can be used to moves the object backward
moveRight(<Real>)	Move the object at the speed specified by the parameter (-100 to 100), a negative parameter value can be used to moves the object left
spin(<Real>)	Spin the object right at the speed specified by the parameter (-100 to 100), a negative parameter value can be to spin the object counterclockwise
stop()	Stops the movement of the object
setSuction(<Real>)	Causes the object to suck the ball with the power specified by the parameter (-100 to 100), a negative parameter value can be used to shoot the ball
getSuction()	Returns the current suction power by the object as a Real number
GPS()	Returns the x and z coordinates of the object as Real numbers
getCompass()	Returns the direction that the object is facing (360 degrees is right, 270 is top, 180 is left and 90 is bottom)

Table 1. Supported commands

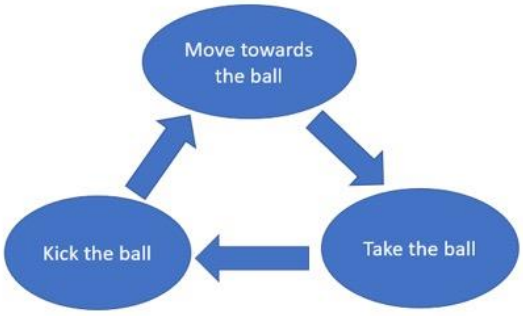

Fig. 2. LOGIN *Plan*

Description

To achieve the simulation in Unity 3D, the project is divided into three parts which is shown in Figure 2 above, (1) Move towards the ball, (2) Take the ball and move the current position with valid angle, (3) Kick the ball. To keep the initial plan functional, our plan needs to have a small sub-plan, implement TCP connection, then analyze and process data received from TCP.

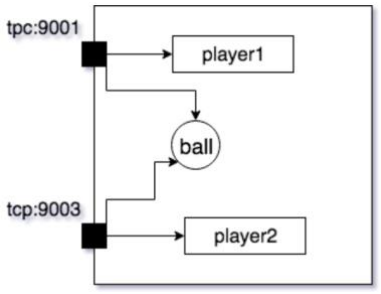

Fig. 3. TCP port connection display

a) Transmission Control Protocol (TCP): Transmission control protocol (TCP) is one of the main protocols in the Internet protocol (IP) suite. It starts with the use of the underlying system, where it complements the Internet protocol (IP). TCP provides a reliable, requested, and error-checking transmission of 8-bit bytes between running applications and over the IP protocol [7]. To keep the server compatible with any programming language support TCP, the Robot Soccer simulation provides no libraries for creating the connection. Instead, there are a sample TCP connection. In our project the TCP connection structure based on blue player java socket program (sample TCP implemented) [4]. We recreated new TCP socket class using Java platform. The communications of the simulation can through Java socket programming TCP. To control the red player, the port (9003) is used to establish the connection and the port (9001) is corresponding to the blue player as shown in Figure 3.

b) Analyze and process data: Specific command/function has a unique string to be received, our mission in this section is to analyze and process those strings, then get valid data set for our project to use. From table-2, GPS command contains each player's location, X and Z stand for the X and Z coordinate respectively. In getCompass command, D is the direction that the object is facing which was mentioned before in table 1. However, there are some situations that we need to deal with when the controller receives invalid data. For example, when player 2 has the ball, if the controller continues to send the GPS command, it will not receive the data we expected instead of "Possession; player2;". Additionally, when the player catches the ball for more than 15 seconds, previous send GPS or getCompass command will not be legal and therefore only return the string "TimeOut; player2". Ultimately, in GPS command there will be a third case. When you send the command (for player2) multiply times, it will receive an abnormal x-coordinate and y-coordinate, without including any player information. Most of the time, the string does not correct the data to match the player 2 location. As the problem mentioned above, we used the Java build-in method try & catch to eliminate the error, then fetched the number in the string and converted it into float type of data. If the string does not contain a number, the function would return an error. Thus, the try & catch would catch the error and go back to the controller class. Although the try & catch method works correctly it is not effective. Sometimes it would negatively impact the controller's function and even cause the entire program to get crash. Therefore, we create another method to process the data, to match the length of the string. Each string has its unique length, to compare the length between the original string and the exact string length we would expect to get. This method will be further discussed in the Evaluation section.

GPS()	1."player2, X, Z;" 2."player1, Z, Z;" 3."X, Z"
getCompass()	1."player2, D;" 2."player1, D;"
Ball Possession	1."Possession; player2;" 2."Possession; player1;"
Time Out	1."TimeOut; player2;" 2."TimeOut; player1;"
Player scoring	1."scored; player2;" 2."scored; player1;"
End of game	"Done;"

Table 2. Messages received from TCP to Controller

We substituted the older plan with an updated one to make the red player move and get the ball. The GPS() command is used to obtain the coordinates of both the ball and player. If the player catches the ball, the player's coordinate should be the same as the ball's coordinate. In order to make the red player always go behind the ball, first obtain the ball coordinates and then create new coordinates for the red player to move by adding 5 to the Z coordinates of the ball. If the ball coordinates are (3,4), we have to move the red player to the coordinates (3,4+5) by calling moveTo(3,4+5). Once the ball is obtained, moveTo() function is ready to move the red player to shoot at the coordinates(-40,0) which is in front of the net. However, after several tests, the solution does not illustrate significant improvement and does not satisfy all the scenarios. For example, if the red player is hit by a blue player, and the angle is not aiming correctly at the ball, the red player will no longer track the ball. And get stuck in the wall. Additionally, if the ball against to the wall, the player will get incorrect coordinates since both of X coordinate, and Z coordinate has the range restriction. Thus, we need to figure out an alternative approach, and the project should to broken down into several steps.

1) ***Move to the ball & Take the ball*:** A big problem entailed by the first approach is the angle. To make a correct angle, an angleSpin function needs to be created, and it is supposed to give stable functionality for the red player to spin at a specific given angle. In the angleSpin function, the current angle is guided by getCompass() command and checks if the current angle satisfies the given condition(angle). The function combine the spin() function and Thread.sleep() command. For example, if the current angle is 180 and the given angle is 200. the current angle is smaller than in given angle, then it starts spinning and the current angle would be updated to 185,190,195,200 and the function stops once the condition is met.

a) Distance formula: For calculating the distance between the ball and the red player. The coordinates of both the ball and the red player are obtained by sending the string message "Ball,GPS" and "Player2,GPS" to Unity 3D environment. The coordinates between the ball and the player are used to determine the distance by using the mathematical distance formula (1)

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

Where (x1, y1) represent player's coordinate and (x2, y2) represent the ball's coordinate. After the distance is calculated, the red player would know how far away the ball is.

b) Cross product & Dot product: In this section, we applied new method to determine direction and angles for the red player using basic geometry theory. Although the robot understands the distance between the ball and itself, it is insufficient to move in the correct direction to track the ball. The ball's position and direction are quite difficult to analyze since the ball could be moving all the time. However, dot product and cross product mathematical theory give a sufficient tool to analyze the ball's position and direction. At the correct state, player and ball should be parallel. Figure 4 represents the correct direction from the red player to the ball.

Step 1. For instance, cross product formula (2)

$$\|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{a}\| \, \|\mathbf{b}\| \, |\sin\theta|. \tag{2}$$

Where *a* represent player's vector, *b* represents ball's vector and θ is the angle between a and b. Since a and b should be parallel, $\sin\theta=0$, so $|a \times b|=0$. For example, (Ax, Ay) represent player's coordinate and (Bx, By) represent the ball's coordinate, the third component Z = Ax * By - Bx * Ay. In order to achieve to correct direction, Z should always be 0.

Step 2. For instance, dot product formula (3)

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta, \tag{3}$$

Where *a* represent player's vector, *b* represents ball's vector and θ is the angle between a and b. From the previous cross product example, *Dot = Ax * Bx + Ay * By*, the Dot helps the red player to determine the side that it needs to spin.
According to the dot product theory, if $\theta < 90$, then Dot > 0, the player should spin to meet the correct angle to move. The combination of the dot product and cross product work sufficiently and the red player is able to locate the direction and correctly get the ball.

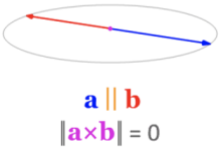

Fig. 4. Cross product to display direction from player to the ball

2) ***Kick the ball*:** Here comes the question "Where is the best location to shoot the ball?" - We assume that the best distance is as close as possible to the net. A loop is set up to check the distance between the red player and the net. In order to make the red player shoot more accurate, the best shooting distance between the player and the net is less than 20. If the distance is greater than 20, the red player keeps moving towards the net by applying the dot and cross product (assume penalty area is a coordinate) as earlier mentioned. After the player reaches the distance less than 20, then it uses setSuction(X) command to shoot the ball. However, to avoid tthe ball bounce back, the range of power X should be between 50 to 60.

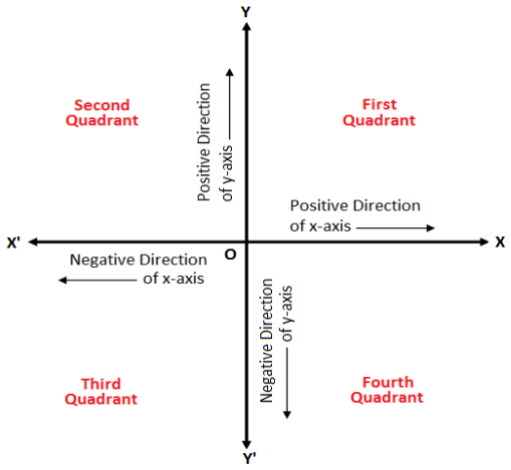

Fig. 5. The Quadrant Scenario

3) ***Stuck problem*:** In certain cases, the red player gets stuck in the wall. In order to eliminate the stuck problem: firstly, we use GPS() command to obtain the X and Z coordinates of the red player and store the coordinates. Secondly, we receive the new coordinates of the red player and check the difference between the coordinates, if the difference is 0, the red player does not move at all and it is stuck somewhere. Thirdly, once the red player is stuck, the controller checks the X coordinate of the red player to see which quadrant it is located. If the X value is greater than 0, it is located at either quadrant 1 or quadrant 4 as shown in the Figure 5 above. Therefore, the controller would move the red player backward and spin to the angle of 180 to keep the red player has enough space to move. Once the X value is less than 0, it is located at either quadrant 2 or quadrant 3. The red player will move backward and spin to the angle of 350. This approach works correctly in the stuck problem. However, the solution may impact the efficiency and performance of the red player since the solution will re-check the message(coordinate) and delay other functionalities.

Evaluation

a) Results: Our main goal is to evaluate the performance of the controller and guarantee that the controller is efficient in solving all the situations we have encountered in the simulation. To ensure the well performance of the controller, several volunteers are invited to test the simulation. In all of our experiments, the controller performs well in tracking the location of the ball, taking the ball and scoring. It works correctly and gets the scores when the controller is executes alone. We also observed that the performance of the controller may depends on the testers' familiarity of the blue player control somehow. The controller works great at the first performance since the testers are new to simulation and they are unfamiliar with the control of the blue player. However, after a few plays, it becomes difficult for the controller to win. The controller does not function as efficiently as the previous runs since the tester gets used to the control of the blue player. Thus, for most of case blue player would win the game. In addition, different platforms would produce various results of the simulation. When the Windows computer performs the simulation, the robot moves smoothly while the Mac computer performs laggy.

b) Problem statement: Our project has encountered a problem of receiving the message(string) through Java TCP. The GPS() command worked in incorrect functionality, it could not obtain the correct string. We thought the cause is due to the Java TCP connection. In order to best express the result of our simulation, a different programming language is also tested such as Python. A similar idea of Java coding has also been implemented into Python. Although the Python version also had the same problem as Java, the Python TCP connection in receiving the message (string) is much faster than Java TCP. Due to the limited time available for this project, unfortunately no best solution would be provided. However, alternated solution would talk in the hardest problem.

c) Hardest Problem: To achieve the best simulation in Unity 3D, our project should eliminate to the incorrect string response. When the red player gets the ball, the "Possession; player2" string will pop up automatically and it will disturb the string receiving. For instance, when GPS() is used to obtain the coordinates of the red player, the string is supposed to be "player2, 48.0000,24.000" but it could appear as "Possession, player2", "score" or "Timeout;player2" after player obtains the ball. Due to the problem, the real coordinates 48.0000 and 24.000 could not be fetched and it gives the error since the input string could not covert to the float type. However, the real cause of the problem could be a string receiving delay. The solution is to check the length of the received string (as early mentioned) since each received string has its own unique length.

(1) Length of the red player GPS() string should between 40 and 46
(2) Length of the ball GPS() string should between 30 and 36
(3) Length of the getCompass() string should between 20 and 25

Once the string length does not satisfy its own string condition, the function GPS() or getCompass() is recalled until the correct string is received. Although the solution works correctly, it may impact the efficiency of the red player since the red player will delay until it gets the correct message (string).

***d) Future discussion**:* Our first step will try to find a better solution to the problem statement. Although there are some other ways to bypass this restriction such as returning an error if the potential is not correct message, it will delay other robot functionalities. However, we will shift our focus to optimize performance, and artificial intelligence would be a new method for our project. Artificial intelligence could help the controller to perform much smarter than the previous hand-coded one. To ensure the AI works properly, our programming language will switch Java to Python. Next, to understand the origin Unity 3D environment code will provide in-depth details to improve the existing solution. In addition, the project also provides the 2 vs. 2 version. Our mission will implement this version of the simulation since it comes with more complex and challenging problems.

***e) Related work**:* The RoboCup organization is a scientific initiative to advance research in artificial intelligence and robotics.[5] The organization has provided a great of

research areas of the artificial intelligence and team strategy, as well as many surprising technological advances. The author presented "BehRobot 3D Soccer Simulation" [6] implemented the motion control of legged robots via artificial intelligence - Recurrent Neural Networks in C++. They have the Behaviors Engine implemented for motors actions, robot path planning, turn camera, walking, and shooting. Our implementation is via TCP to obtain the GPS strings and apply geometry theory for all the actions of robots. They made the robot in the real 3D environment and performed the soccer game, whereas our simulation was in a virtual 3D unity environment. Both projects have similar ideas of motion control: firstly, move to the ball, secondly targeting, and then follow by shooting the ball. Moreover, "BehRobot 3D Soccer Simulation" has multiple robots in the simulation, whereas our project is 1 vs 1 robot version of the simulation.

Conclusion

In this paper, we presented and discussed different designs and implementation methods for creating automatic robots in the Unity 3D soccer simulation environment with a TCP connection from controller to Unity. We propose a basic automatic controller based on the combination of the cross product and dot product to enable the red player to perform and interact with the ball and blue robot. We implemented our approach in the Java platform and evaluated the performance of the controller. Our experiments showed that the red robot worked correctly and got the scores when the controller is executed alone. In addition, the different platforms would produce various results of the simulation. The Windows system performs the robot moves smoothly while in Mac system performs laggy. Our future work will focus on optimizing the performance of the controller and try to make the robot more stable and reliable as the result of our researches.

Reference

[1] Gabel, T. (n.d.). Robotic Soccer Simulation. [online]. Retrieved from http://tgabel.de/cms/research/multi-agent-systems/rss/

[2] Martinovic, J. Improving Strategy in Robot Soccer Game by Sequence Extraction[online]. Retrieved from https://www.researchgate.net/publication/275541617_Improving_Strategy_in_Robot_Soccer_Game_by_Sequence_Extraction

[3] The Robocup Federation. The robocup objective [online]. Retrieved from https://www.robocup.org/objective

[4] MDETools 2019, Overview of the Challenge Problem [online]. Retrieved from https://mdetools.github.io/mdetools19/challengeproblem.html

[5] The Robocup Federation. About robocup [online]. Retrieved from https://www.robocup.org/a_brief_history_of_robocup

[6] Ali Ahmadi, Ehsan Shahri, Majid Delshad.2017.BehRobot 3D Soccer Simulation. Institute of Robotics and Intelligent Systems. Islamic Azad University of Isfahan, Isfahan, Iran.

[7] Wikipedia. 2020. WikipediA: Transmission Control Protocol. Retrieved from https://en.wikipedia.org/wiki/Transmission_Control_Protocol