MOBILE ROBOT DECISION MAKING USING BEHAVIOUR TREE

Arnel D. Zamayla, Marc-Eduard E. Navarro, Ingvar R. Estorco Jr.
Electrical Electronics and Computer Engineering Department
MSU-Iligan Institute of Technology Iligan City, Philippines

ABSTRACT

In this study, we used the BT as a mobile decision-making system for mobile robots to investigate whether the proposed system is accurate, and whether there is a significant difference in the performance of the system in the simulation and in actual environment. A robot soccer system was used as a test bed for this study to confirm the accuracy of the execution of the behaviours and task performed in both ideal and experimental environments. The method consists of behaviours suited for different cases. The experiment includes success rate measurement on the execution of the behaviours as well as the task completion in both actual and ideal environment. The proposed method was able to execute the behaviours with a 100% accuracy rate for positioning and defense behaviours and a 60% accuracy rate for shooting and obstacle avoidance behaviours. The task completion which involves the presence of opposing team robots, it has an average of 75% success rate achieved in simulated test while an average of 41% success rate achieved in the actual environmental test. The robot was able perform accurately the behavior formulated. The results also show a significant difference in the performance rate between the simulation and the actual experiment.

Keywords: *Behaviour Trees (BT); Soccer Robot; test bed;*

Contact Information: arnel.zamayla@g.msuiit.edu.ph

INTRODUCTION

Robotics is all around us, from everyday cars to autonomous Mars rovers. We need robotic assistance to stretch the limits of what can be done: think of production speed; ease and comfort of living; work in hazardous environments; precision tasks in surgery; and the list goes on ("RoboCup: not just fun and games," 2014). Mobile robots are considered the most representative and most amazing category of Mechatronic systems in which they can perform real world tasks with difficult terrains as an example (Mihai, 2014). Designing effective behaviour to complete complex tasks for small robotic platforms is a major challenge. Small robots with limited computational capabilities are becoming more common due to their ability to swarm and achieve a task collaboratively (Scheper, 2014). Behaviour Trees (BTs) offer one such candidate architecture which could prove a promising foundation for reliable behavioural systems. The BT concept emerged from the computer gaming industry, where they were first described as powerful tools to provide intelligent behaviours for non-player characters in high profile computer games (Olsson, 2016).

A robot soccer system is widely used to incorporate ideas involving artificial intelligence. The opposing team within the game has the primary objective to win. The game presents difficulties to be considered such as finding a suitable position for shooting; navigation through the environment given its location and the location of other robots; and role switching; among the teammates that will handle the ball.

To answer these problems, a goal-oriented AI approach is needed. Finite state machines or Markov models are some of the approaches in robot control behavior. Recently a new approach uses the behavior tree for robot control.

A behavior tree is a hierarchical finite state machine with custom nodes and leaf. This leaf can be an action or a condition. To aid in the decision-making process, the execution of a behavior tree controls the robot behavior (Luca Siqueira & Pieri, 2014).

Behaviour Tree

Behavior trees (BT) have been proposed as a new approach to the designing of game AI. Their advantages over the traditional AI approach lie in their simplicity to design and to implement, it is relatively scalable when games get larger and more complex, and because of the modular approach, reusability and portability are the least of the problems. Behavior trees provide a way to define intelligent control. Their key characteristics include their being simple to define, their scalability to exhibit complex AI and their being modular for reusability. Behavior trees are goal oriented, with each tree associated together, allowing the implementation of complex behaviors by first defining smaller, sub-behaviors (Lim, 2009). The use of behavior trees simplifies the decision-making process. To address the growing mess of state transitions of FSMs, behaviour trees use a more restrictive and more structured traversal approach. BTs encounters less difficulty in defining complex states and it is very easy to see the logic in it. They are fast to execute and easy to maintain. (Abiyev et al., 2016)

Behavior Trees are formulated as directed graphs with a tree structure. The topmost node, denoted as the root node, has one or more child nodes, which, in turn, can also have children. Nodes which have children are sometimes referred to as composition nodes. A child node with no children of its own is called a leaf. The root node is the only node without a parent. Each node in the tree belongs to one of the six types of nodes shown in Table I. For non-leaf nodes, they can be any of the following four types: Selector, Sequence, Decorator, or Parallel. Leaf nodes come in the form of either Action or Condition. The execution of a BT starts at the root node every time and then progressively traverses the structure in a depth-first fashion, polling (usually referred to as ticking) every node as the execution proceeds down and up the tree. When a tick reaches a leaf node, some computation or action is performed, and then returns as Success, Failure, or Running. The return status propagates up the tree eventually back to the root node. Executing BT immediately again will poll the same nodes, even though they returned success the last time. There is no memory of the last execution or skipping of previously successful or failed nodes in a rudimentary BT. A selector node will begin to tick its children in order. If the first child fails, the execution continues with the following child and it is ticked. If a node fails, it goes to the next one as a fallback; hence, the selector node is sometimes called Fallback-node. If a child succeeds, the selector also returns success and does not move on to the following children.

A sequence node ticks its children in sequence, as it tries to ensure that a number of sequential tasks are all performed. If any child returns failure, the sequence has failed and it will propagate up. The sequence node only returns success if all children succeed. A parallel node ticks all its children at the same time, allowing several Action nodes to enter a running state at the same time. The requirement for how many children need to succeed before the Parallel node itself reports success/failure can be customized on a per-instance basis. The decorator node wraps the functionality of the underlying child or sub tree.

It can influence the behavior of underlying node/s or modify the return state. A typical kind of Decorator node is an inverter, which flips the return status of its child from Success to Failure and vice versa. Another example of Decorator setting a time limit on the execution of its child occurs by breaking and returning failure, if the child did not complete fast enough. The Condition node is analogous to a simple if-statement. If the conditional check is true, the node returns Success and Failure, if false.

A condition node will never return a Running status.

Table 1. Node Types

Node Type	Succeeds	Fails	Running
Selector	If one child succeeds	If all children fail	If one child is running
Sequence	If all children succeed	If one child fails	If one child is running
Decorator	Varies	Varies	Varies
Parallel	If N children succeed	If M-N children succeed	If all children are running
Action	Upon completion	When impossible to complete	During completion
Condition	If True	If False	Never

Method

The decision making system is the one that assesses the situation and selects the appropriate actions that the soccer robot has to make. This study introduces a behavior tree which allows a generic role for each soccer robot, so that each can switch between attacking and defending. The soccer robot uses a direct path in navigation in order to get to its destination faster. An obstacle avoidance is implemented in order to evade obstacles along the way.

Robot Movement

A direction position is designated by the decision making system, depending on the executed behavior. Based on Fig. 1, the angle of the direction position is determined using the formula:

$$\theta_d = \tan^{-1}\frac{(y_d - y)}{(x_d - x)} \qquad \textit{Eq 1}$$

The angle error θ_e is determined by subtracting the direction angle from the current heading angle. The mobile robot used in this paper uses a drive mechanism known as differential drive. It consists of two drive wheels mounted on a common axis, with each wheel independently being driven either forward or backward. The velocity of each wheel is varied, depending on the angle error with respect to the desired heading. To calculate the velocity of each wheel this is used:

$$V_L = V_C S(de) + K_p\theta_e \qquad \textit{Eq2}$$

$$V_R = V_C S(de) - K_p\theta_e \qquad \textit{Eq3}$$

V_C is the desired maximum average velocity of the mobile robot. V_C is then multiplied by the sigmoid function $S(t) = \frac{1}{1-e^{-t}}$ to vary the velocity, depending on distance *de* which is the distance from the robot to the direction position. K_p is a proportional gain which is set differently for different angle errors. This is due to the relationship between angle errors and velocity. If smaller K_p is used for bigger angle errors and also for small angle error, the velocity values sent to the robot are smaller than the required values.

Figure 1. Robot Positioning

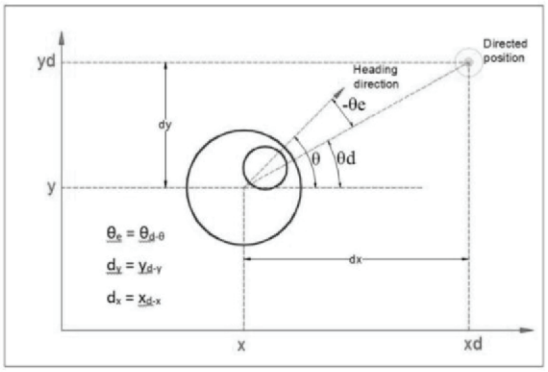
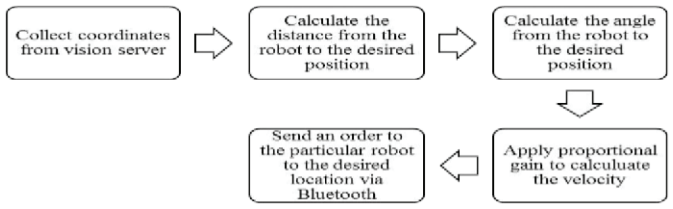


Figure 2. Position Process Chart


Collect coordinates from vision server
Calculate the distance from the robot to the desired position
Calculate the angle from the robot to the desired position
Apply proportional gain to calculuate the velocity
Send an order to the particular robot to the desired location via Bluetooth

Obstacle Avoidance

Using the coordinates that can be collected from the vision server, we can emulate a sensor for the obstacle detection of the mobile robot. By extending a point from the robot's position by a desired distance clearance and angle, it can act as a sensor. By assigning a sensor point at the left, front and right side of the mobile robot, it can determine if there is an obstacle ahead of it. An obstacle is represented as circles with radius more than the actual radius for collision clearance purposes. If it circumscribes a sensor point, it would trigger the mobile robot to steer left or right depending on which sensor point is involved. Fig. 3. shows the sensors of the robot used for obstacle avoidance. Table 2 shows how the velocity of the left and right wheels react when the sensors are circumscribed by the obstacle.

Figure 3. Obstacle Avoidance Sensors.

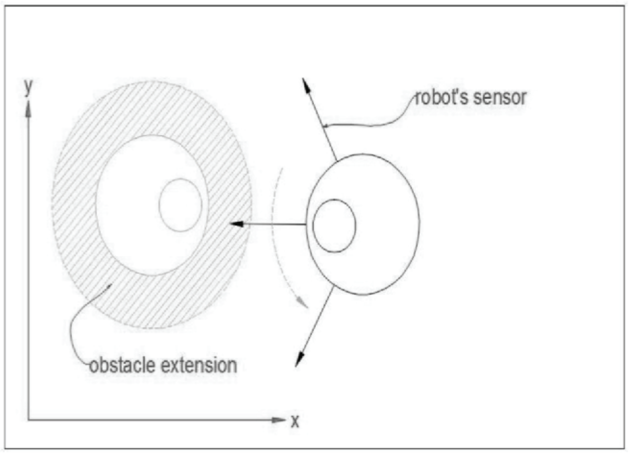
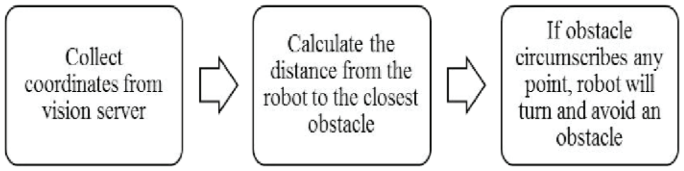


Table 2. Node Types

Left	Front	Right	Left Velocity ($V_L > 0$)	Right Velocity ($V_R > 0$)
0	0	0	V_L	V_R
0	0	1	$V_L/2$	V_R
0	1	0	$V_L/2$	$V_R/2$
0	1	1	$V_L/3$	V_R
1	0	0	V_L	$V_R/2$
1	0	1	$V_L/2$	$V_R/2$
1	1	0	V_L	$V_R/3$
1	1	1	$-V_L$	$-V_R$

In this paper, all mobile robots were considered an obstacle aside from itself, and the ball was considered an obstacle when the ball is approached from a direction leading home side of the field in order to prevent the home team robots to push the ball toward the home goal.

Figure 4. Obstacle Avoidance Process Chart


Collect coordinates from vision server
Calculate the distance from the robot to the closest obstacle
If obstacle circumscribes any point, robot will turn and avoid an obstacle

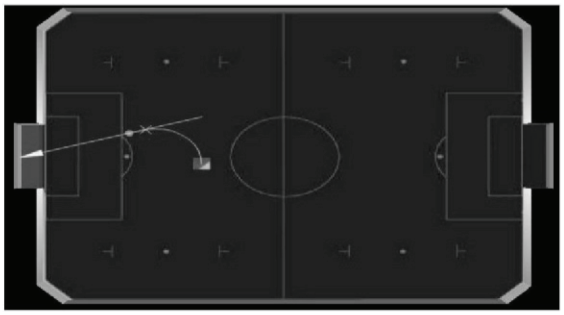
Shooting

Scoring points is the main goal of a soccer game in order to win a game. In order to shoot the ball the assigned player must approach the ball and push the ball into the goal. First the player must align itself with respect to the ball and the goal. Given the coordinates of the ball and the goal, we can generate an equation using the point-slope equation of a line.

$$y - y_1 = m(x - x_1) \quad \textbf{Eq 4}$$

Using the equation of the line we can pick a point within the line which is behind the ball and send a move command to the player. If the player is within or near the shooting point, it can then be checked if it is possible to push the ball to the goal by checking if the ray originating from the players' orientation passes through the line segment of the goal line.

Figure 5. Shooting Point Positioning



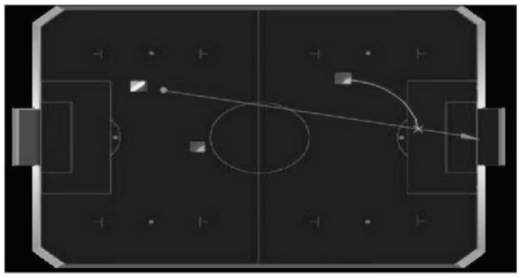
In Fig. 5, the attacker locates a suitable position where it will line itself with the ball and the goal. In the figure below, the mobile robot checks if it is possible to push the ball towards the goal and can probably score a point.

Defending

For defense, given the coordinates of the ball and the home goal, we can calculate for a position between the ball and the home goal, using the midpoint formula.

Figure 6. Defense Positioning

$$\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right) \qquad \text{Eq 5}$$

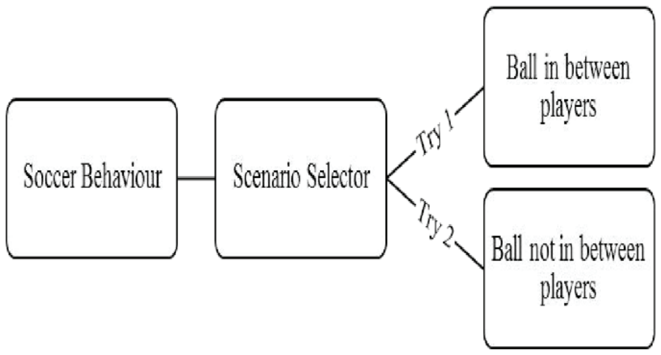


In Fig. 6, the defender locates a defending position, which is between the goal and the ball, and immediately moves to the specified position.

Behaviour Tree

In order for the decision making system to determine which mobile robot will approach the ball, the ball placement with respect to the mobile robots is needed. In the figure below, the behavior tree checks if the ball is situated between the mobile robots or is not between the players.

Figure 7. Soccer Behaviour



Ball In Between Player Scenario

In the event that the ball is situated between the players, the player who is nearer the goal will fall back and attempt to defend, while the other player will approach the ball facing the goal. In the figure below, the behavior tree traverses first the attack sequence and checks if the teammate robot is nearer the goal; if so, the current robot fetches the ball. In the situation that the Attack Sequence fails the current robot will defend.

Figure 8. The Sub-tree in Between Player Scenario

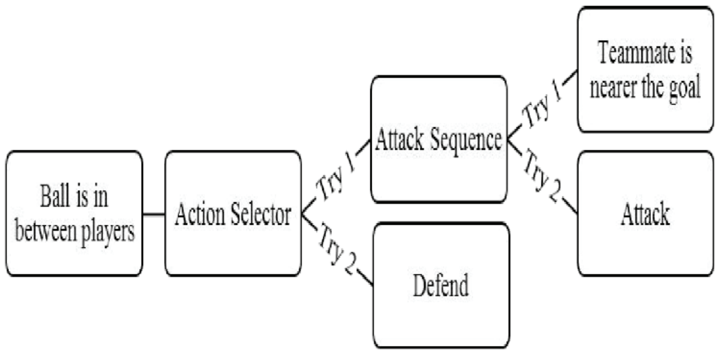
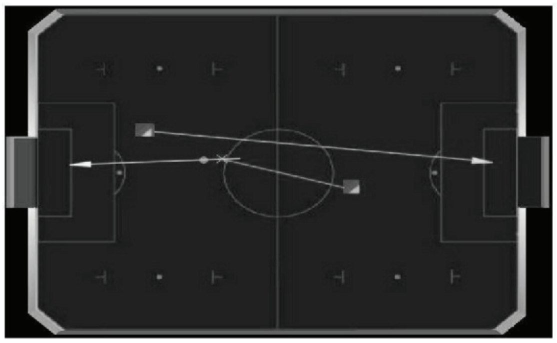


Figure 9. Example of Ball in Between Player Scenario



Ball Not In Between Player Scenario

In the event the ball is not in between players, the player nearest the ball approaches the ball and the other player will fall back and defend. In the figure below, the tree traverses the Defense Sequence and checks if the teammate robot is nearer the ball, then defends; otherwise, the tree will execute the attack action node and become the attacker.

Figure 10. The Sub-tree for Ball not in Between Player Scenario

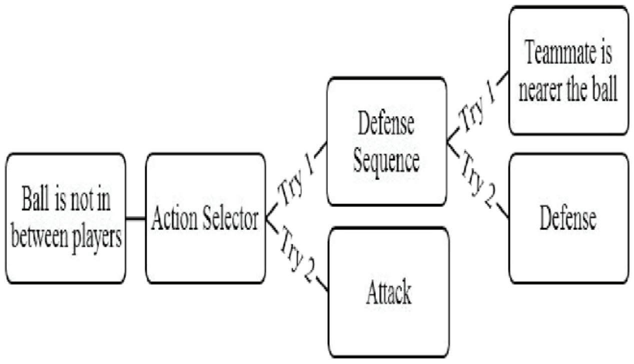


Figure 11. Example Scenario When the Ball is in the Left Side of the Field

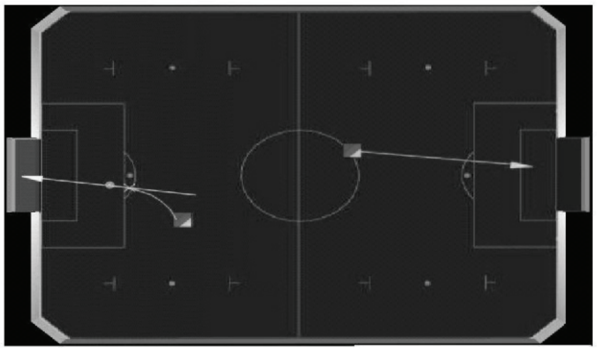
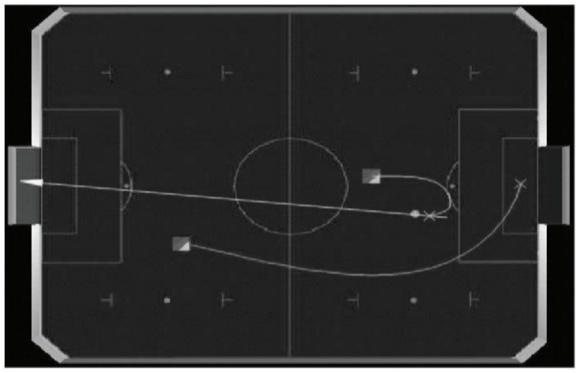


Figure 12. Example Scenario When the Ball is in the Right Side of the Field



RESULTS

The experiment for the ideal scenarios has been conducted on a computer with 1.90 GHz Intel Celeron, using 3D Robot Simulator developed by RSS Development team. The experiment for the actual scenarios has been conducted using two-wheeled mobile robots equipped with Arduino UNO. A 1.90 GHz Intel Celeron host computer was used to run the algorithm that contains all of the behaviours. Each robot wears a unique color patch on its top and an overhead CMOS camera monitors them. This is connected to a host computer that broadcasts the information through hotspot from an object tracking system. The commands sent to the robot are via the Bluetooth from a computer host. Table 3 shows the summary of the test results of the accuracy of the actual mobile robots in executing different behaviours as it respond to scenarios defined in the previous chapter.

Table 3. Test Result of Behavioural System on Actual Scenarios

Behaviour	Trial Total	Success Rate
Case 1: Positioning	10	100%
Case 2: Defending	10	100%
Case 3: Ball is in-between I	10	100%
Case 4: Ball is in-between II	10	100%
Case 5: Ball is not in-between I	10	100%
Case 6: Ball is not in-between II	10	100%
Case 7: Shooting	10	60%
Case 8: Obstacle avoidance	10	60%

The task completion rate of behavioural system on simulation has an average of 75% with a standard deviation of 0.44; on the actual robots, it has an average of 41% with a standard deviation of 0.41. The mean of simulation minus actual is 34%. The computation shows a T-computed value of 4.3902 which is higher than the t-critical value of 1.984 with a degree of freedom equals to 143 and a significance level of 0.05. Table 4 shows the summary of the unpaired T-test result of the task completion rate between the simulation and actual experiments.

Table 4. Unpaired T Test Result

	1	2	df	T-value	T-Critical	Two tailed P-value
Simulation	0.75	0.44	143	1.984	1.984	<0.0001
Actual	0.41	0.49				

CONCLUSION

This paper presented the Behaviour Tree Method for mobile robot decision making. The accuracy of the behavioural system in executing different behaviours was confirmed through an experiment. Furthermore, the accuracy rate of the task completion potential of the robots using the proposed system was also presented. There was a high significant difference between the ideal and actual experiments.

In the actual field, the mobile robots were able to perform the required skills for soccer robot though they usually exaggerate performing the actions. These exaggerations are caused by the accumulated delays of data transmission and processing time needed for the robot to execute the commands given. The robots also find it difficult to navigate and acquire the ball if the ball is located in the corner of the field. This is due to the friction present between the robot and the field's wall.

The shortcoming of the algorithm lies upon not taking into consideration the robots' performance. The results showed that the performance rate of the actual robots was low compared to the performance rate of the simulation. Therefore, a robot decision making that can accommodate these diverse factors needs to be studied. Moreover, the shooting capability and the navigation system need further improvement to achieve more satisfying results.

ACKNOWLEDGMENT

The researchers would like to thank the Faculty of the Department of Electrical, Electronics, and Computer Engineering of MSU-Iligan Institute of Technology for providing the facilities for this research.

REFERENCES

Abiyev, R. Abizada, S. Akkaya, N. Aytac, E. Cagman, A. and Gunsel, I. 2016. *Robot Soccer Control Using Behavior Trees and Fuzzy Logic.* 12th International Conference on Application of Fuzzy Systems and Soft Computing, ICAFS 2016

Lim, C. 2009. *An A.I. Player for DEFCON: An Evolutionary Approach Using Behavior Trees.* Retrieved from Imperial College London website: http://www.doc.ic.ac.uk/teaching/distinguishedprojects/2009/c.lim.pdf

Luca Siqueira, F. and Pieri, R. 2014. *A context-aware approach to the navigation of mobile robots.* The 5th Conference Information, Intelligence, Systems and Applications, IISA 2014. 10.1109/IISA.2014.6878733

Mihai, D. 2014. *Wheeled Mobile Robot Development Platforms – From Budget to Full Featured.* Retrieved from https://www.smashingrobotics.com/wheeled-mobile-robot-developmentplatforms-from-budget-to-full-featured/

Olsson, M. 2016. *Behavior Trees for decision-making in Autonomous Driving.* Examensarbete I Datateknik 300hp, Avancerad Niva.

"RoboCup: not just fun and games.". 2014. Retrieved from http://www.technologist.eu/robocup-not-just-fun-and-games/

Scheper, K. 2014. *Behaviour Trees for Evolutionary Robotics Reducing the Reality Gap.*