Object Detection for Robot Coordination in Robotics Soccer

C. O. Yinka-Banjo*, O. A. Ugot, E. Ehiorobo

Department of Computer Science, University of Lagos, Akoka, Nigeria.

ABSTRACT: In June 2018, iCog Labs held its second annual robosoccer competition which featured groups of humanoid robots playing soccer against each other. The authors were members of a team called upon to represent Nigeria with the University of Lagos at the competition which took place in Ethiopia. The work here presents a review of the approach taken to address the problem of automating robot coordination in real-world soccer applications. The design methodology relies on the Robot Operating System (ROS) as the platform upon which an asynchronous communication network between each robot and a central server is built. On the network, each robot is a node that consists of sub nodes for object detection and motion control. For object detection the work makes use of the you only look once (YOLO)v2 deep learning algorithm, and a simple decision-making algorithm for controlling vcv the robot based on the objects detected is devised. To quantify the object detection results, the common objects in context (COCO) evaluation metric is used. The results indicate an average recall and precision of 84% across different IOU. For qualitative results on the robot coordination in the ball's direction, a reference to the open-source implementation of the work has been provided.

KEYWORDS: Artificial Neural Networks, Deep learning, Multi-agent, Object detection, Robotics.

[Received Sep. 19, 2021; Revised Feb. 25, 2022; Accepted Mar. 15, 2022] Print ISSN: 0189-9546 | Online ISSN: 2437-2110

I. INTRODUCTION

The 2018 iCog Makers Robo-Soccer Competition is one of a series of competitions held by the iCog Labs in Ethiopia with the aim of encouraging Africans to work towards the RoboCup goal of making robots capable of playing soccer against the FIFA 2050 World Cup human champions (Ferrein and Steinbauer, 2016). The competition (iCog Labs, 2018) required that participating groups hack RoboSapiens – human-like toys designed by Mark W. Tilden – and by incorporating artificial intelligence, teach the robots to play soccer autonomously.

The competition setup (iCog Labs, 2018) had teams from 5 universities in Ethiopia, Kenya and Nigeria playing against one another. Each match had two teams of two robots each play against each other. The soccer pitch was artificial and green in colour with colour-coded posts and markers around the field. The ball was also coloured red for easy identification by robots. A game winner was decided by the number of goals scored in the match. If the match ended in a draw, then penalties were played without goalkeepers.

For the hardware, the work makes use of the WowWee RoboSapien X, a bi-pedal robot capable of a range of human-like actions like dancing, walking and other neck, arm or waist motions. The robot was designed as a toy by the WowWee Group Ltd. The robot was hacked and a Raspberry Pi 3 board was inserted for autonomous control of the robot, a Raspberry Pi camera for vision, and a servo motor to allow the robot to turn its camera around for a wider range of sight. For the software, the Robot Operating System (ROS) is used, an abstraction layer which provides libraries to handle communication among the robots.

Recent work in robot coordination revolves around the use of reinforcement learning for training the robots in a simulated environment. In such studies, robot coordination relies on a reinforcement learning algorithm to make the right decision during a coordination task (Wang *et al*, 2020). Other studies address the coordination problem starting from the design and modelling phase of the robot, where consideration such as the anatomy of the robot is taken into consideration (Spensieri *et al*, 2021).

The work presented in this paper approaches the problem by training the robots to detect objects in their environment, and communicate the detected objects to a *master* node. This master node is then tasked with directing their movements towards the detected object. After collating the inputs from the different robots, the master node would decide on what each robot should do as its next step. The communication between the master nodes and the robots was carried out using ROS.

This paper describes the design, the implementation and results during the iCog Labs RoboSoccer competition. The paper also includes problems that were faced during implementation and suggest approaches to solve them.

II. LITERATURE REVIEW

A) Robot Coordination

While a lot of consideration must be put into the design of the individual humanoid robot in order to make it autonomous and human-like, the robots also must be designed in such a way that they are able to communicate, as soccer is a

*Corresponding author: cyinkabanjo@unilag.edu.ng

doi: http://dx.doi.org/10.4314/njtd.v19i2.5

team game. This coordination might include passing, avoiding collision and electing who should go for the ball. Achieving coordination among robot players can be done by making them able to communicate over a network as well as visually by placing markers on teammate robots for identification (Xin *et al*, 2020).

B) Deep learning

Deep learning is a subset of Machine learning (Lecun *et al*, 2015), that involves the design of artificial neural networks that are stacked in hierarchical layers. The "deep" in deep learning highlights the need to stack these neural network layers into many layers. Deep learning architectures have proven to be the state-of-the-art in computer vision tasks (such as object detection and classification) and natural language processing. An artificial neural network architecture consists of an input, a set of hidden layers and output (see Figure 1). Each hidden layer consists of weights that define output of the neural network (Zhang *et al*, 2021).

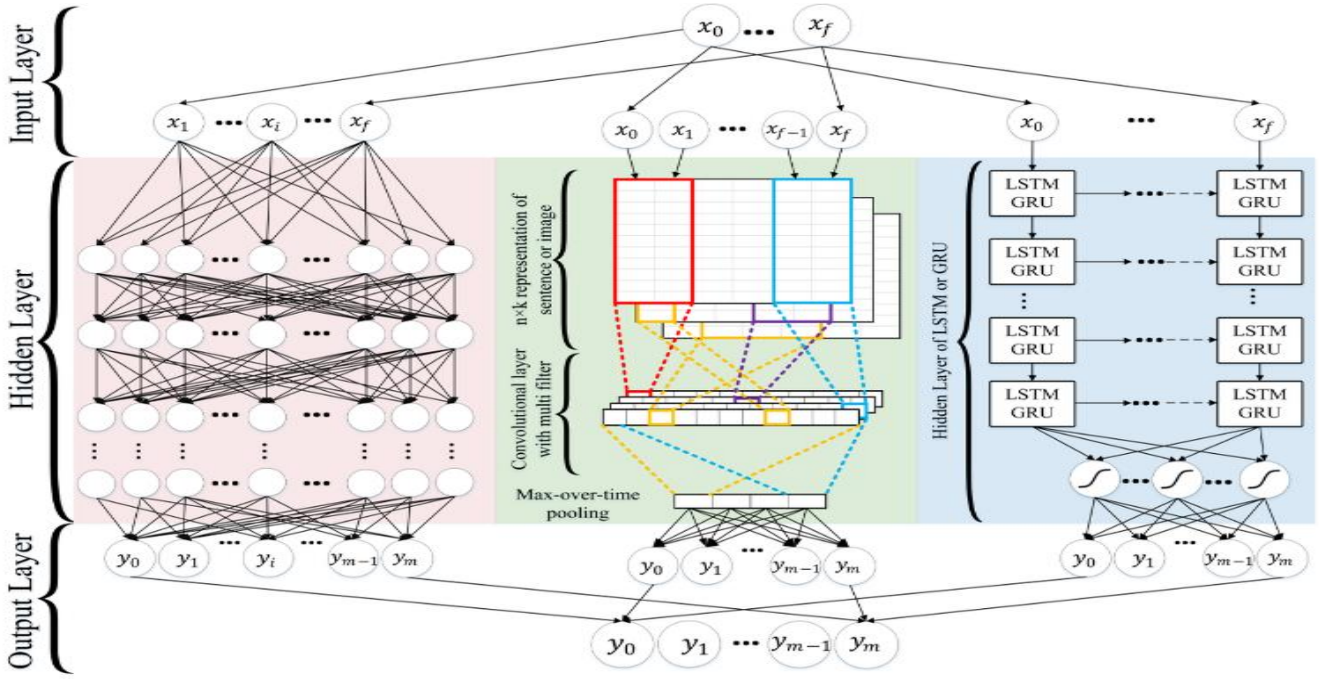

Figure 1: Architecture of an artificial neural network showing various architectures including the convolutional neural network and recurrent neural networks.

One can summarize the working of a neural network with Eqn. (1).

$$y = xW^T + b \quad (1)$$

In Eqn. (1), *x* is the input vector, *W* is the weight vector from the hidden layers, b is the bias needed to normalize the output *y*. Eqn. (1) describes the forward propagation of the neural network, the process of propagating the input through the neural network to get the output. One can view the neural network as a large function approximator that maps the input *x* to an output *y*. The goal, therefore, is to adjust the weights of the neural network so that it can correctly approximate the output y, given the input x. The backpropagation (backprop) algorithm provides a means to do this (Lecun *et al* 1988), one can summarize backprop with Eqns. 2, 3 and 4 below

$$e = g(xW^T + b) \quad (2)$$

$$\frac{\partial e}{\partial x} = \frac{\partial e}{\partial y}\frac{dy}{dW} \quad (3)$$

$$w_i \leftarrow w_i - \alpha \frac{\partial e}{\partial x} Loss(w_i) \quad (4)$$

Using Eqn. (1), one can calculate the predicted output y. Then, using Eqn. (2), one can calculate the loss which gives us an empirical means of measuring how different the predicted output is from the ground truth. In Eqn. (2), the cost function g(y) enables us to compute the loss e. In Eqn. (3), one can compute the proportion to which each parameter of our neural network in Eqn. (1), contributes to the overall loss e. This process is achieved (using the chain rule of calculus) through the backprop algorithm where one can calculate the partial derivative of the loss with respect to the neural network parameters in Eqn. (3).

Finally, it is important to have a means by which one can adjust our neural network parameters once the proportion to which they contribute to the loss e has been found. One can use an optimization algorithm such as gradient descent to adjust the neural network parameters (Schmidt *et al*, 2021). In Eqn. (4), the weights of the neural network are adjusted by subtracting the partial derivative from Eqn. (3) from the current weight value.

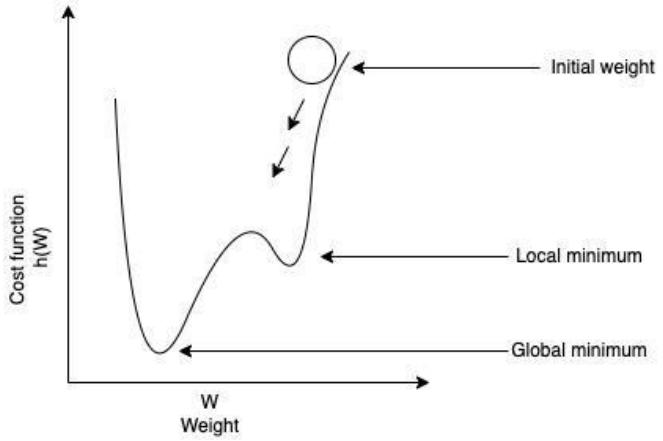

Figure 2: Illustrating Gradient descent.

C. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural network architecture that makes use of the convolution operator in its hidden layers (Lindsay, 2021). It takes advantage of the fact that the input consists of 2-dimensional data structure, and they constrain the architecture in a more sensible way. Unlike a regular Neural Network, the layers of a CNN have neurons arranged in 3 dimensions: width, height, depth. The word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network. In general, a CNN consists of 3 layers.

1) Convolutional

Convolutional layers consist of a rectangular grid of neurons. It requires that the previous layer also be a rectangular grid of neurons. Each neuron takes inputs from a rectangular section of the previous layer; the weights for this rectangular section are the same for each neuron in the convolutional layer. Thus, the convolutional layer is an image convolution of the previous layer, where the weights specify the convolution filter (Zhou, 2020).

In addition, there may be several grids in each convolutional layer; each grid takes inputs from all the grids in the previous layer, using potentially different filters.

2) Max-pooling

After each convolutional layer, there may be a pooling layer. The pooling layer takes small rectangular blocks from the convolutional layer and subsamples it to produce a single output from that block. There are several ways to do this pooling, such as taking the average or the maximum, or a learned linear combination of the neurons in the block.

3) Fully connected

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. A fully connected layer takes all neurons in the previous layer (be it fully connected, pooling, or convolutional) and connects it to every single neuron it has. Fully connected layers are not spatially located anymore, so there can be no convolutional layers after a fully connected layer.

D. You Only Look Once (YOLO) Object detection

YOLO is an object detection neural network architecture that is trained to identify viewpoint objects in an image (Redmon *et al*, 2016). The basic idea behind YOLO (Figure 3), is that the input image is divided into an S x S grid of cells. For each object that is present on the image, one grid cell is said to be "responsible" for predicting it. That is the cell where the center of the object falls into.

Each grid cell predicts B bounding boxes as well as C class probabilities. The bounding box prediction has 5 components: (x, y, w, h, confidence). The (x, y) coordinates represent the center of the box, relative to the grid cell. These coordinates are normalized to fall between 0 and 1. The (w, h) box dimensions are also normalized to [0, 1], relative to the image size. It is also necessary to predict the class probabilities, $Pr(Class(i) \mid Object)$. This probability is conditioned on the grid cell containing one object.

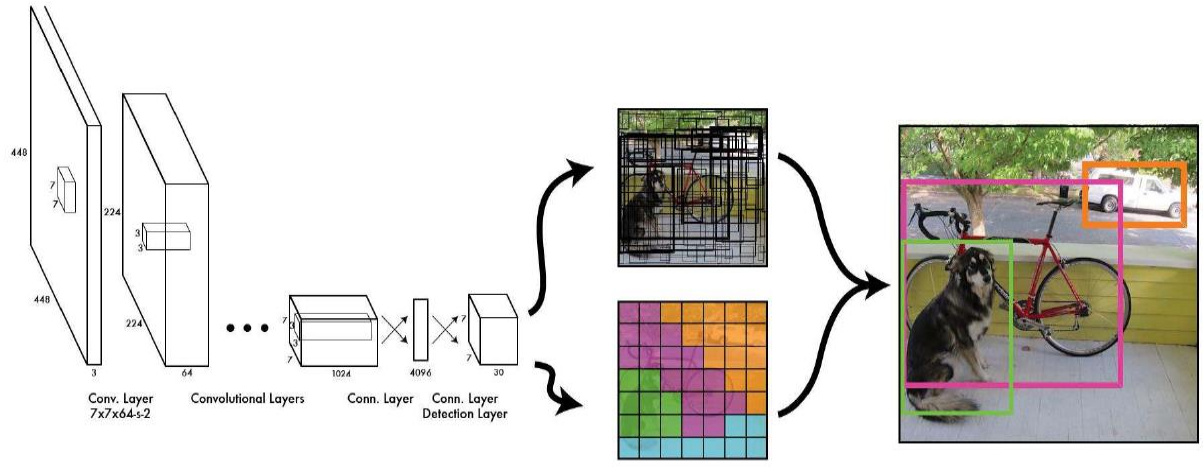

Figure 3: An illustration of how the YOLO algorithm works (Redmon *et al*, 2016).

III. METHODS

The process of creating the robots for the iCog Labs RoboSoccer competition involved hacking into 3 WowWee Robosapiens X robots. The hardware setup (see Figure 4) for each included a Raspberry Pi 3 computer, 2 Raspberry Pi cameras (top and bottom), a power bank for power supply, a servo motor for turning the camera around to give a wider range of view, and a breadboard and jumper cables.

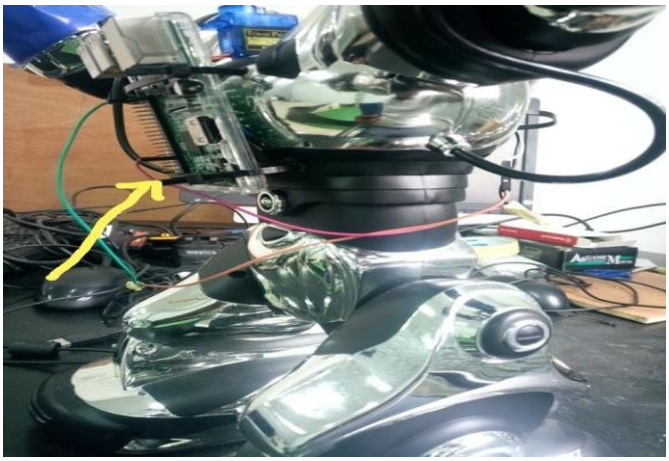

Figure 4: The hacking process involved bypassing the in-built Robosapiens microcontroller (shown in the image) and installing our own raspberry pi controller.

The software setup involved installing the Raspbian OS on the Raspberry Pi computer on each robot, connecting them to a Wi-Fi network and installing ROS to handle networking. The ROS package was set up on a laptop to handle communication between the laptop and the robots. The following subsections discuss other components of our system in the following sections.

A. Dataset Preparation

To train an object detector to detect the ball and the goal post, a large dataset is required to train the YOLO model. Steps taken in the dataset preparation includes:

1) Gathering the images

The images for the dataset must be realistic with respect to the robot's view of the ball and goal post. Therefore, to gather these images, multiple snapshots of the objects was taken using the camera strapped to the robot. This was done while controlling the robots to simulate a typical gameplay. As the robots moved around, the camera took snapshots of the objects on the field. Figure 5 shows examples of some of the gathered images.

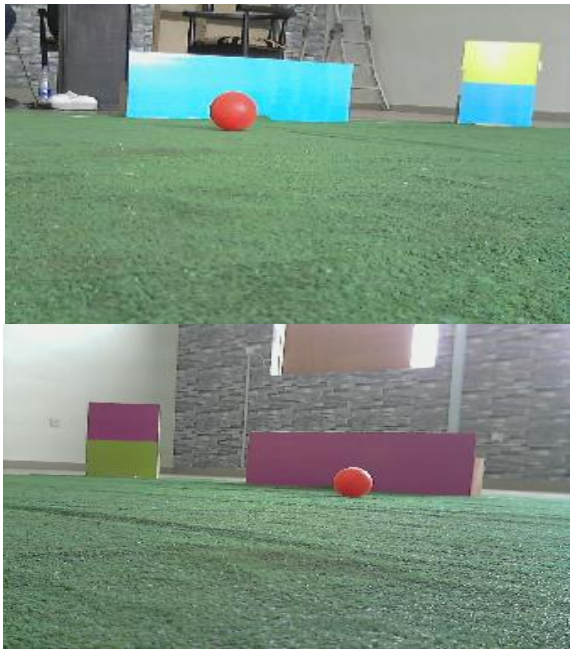

Figure 5: Sample images gathered from the robot's camera.

2) Preprocessing

The preprocessing task is executed on the gathered images. The YOLO algorithm requires an XML file describing the coordinates of the desired object in the image. Thus, each image in the dataset was formatted using an XML file describing the coordinates of the ball and goal post in the image. The total dataset contained 3000 images. See figure 6 for results from the object detection training.

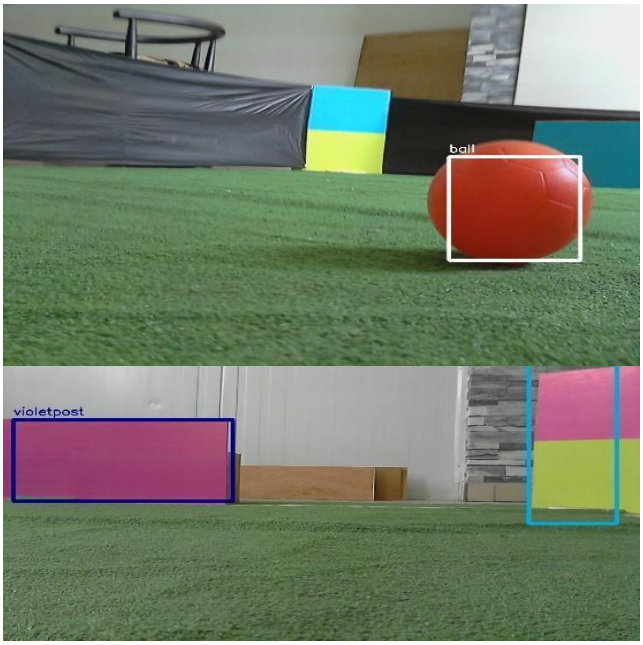

Figure 6: Sample results from our object detection module for detecting the ball and goal post.

B. Object Detection Module

The task for the object detection module is to identify the objects that are present in an image frame. With respect to the problem of robot soccer, this means that there is a need to be able to identify when a ball or a goal post is in an image frame. This task is non-trivial and involves a tedious process of preparing a training dataset, then training a deep neural network to perform the task.

Algorithm 1 Object Detection Module

Require: *imageFrame*

1: *detectedObjects* ← *yolov2.detect*(*imageFrame*)
2: *boundingBox* ← [] ▷ bounding box coordinates
3: **for** *object* in *detectedObjects* **do**
4: **if** *object.get*("*label*") in ["*ball*","*post*"] and *object.get*("*confidence*") > 0.3 **then**
5: *Label* ← *object.get*("*label*")
6: *topLeft* ← *object.get*("*topleft*")
7: *bottomRight* ← *object.get*("*bottomright*")
8: *boundingBox* ← ["*topLeft*" : (*topLeft.get*("*x*"),
9: *topLeft.get*("*y*")),"*bottomRight*" : (*bottomRight.get*("*x*"),
10: *bottomRight.get*("*y*")),"*label*" : *Label*]
11: **end if**
12: **end for**
13: return *boundingBox*

Figure 7: An Algorithm describing the Object detection Module.

The primary object detection technique used is the YOLOV2 algorithm (Redmon *et al*, 2017), algorithm 1 outlines how YOLOv2 was used for the custom ball and goal post detection algorithm.

In line 1 the compressed image frame is read and a call to the YOLOv2 object detection class is made in order to detect the objects in the image. Line 4 checks if the label for the object is "ball" or "post", and the confidence is greater than 0.3. If line 4 is true, then the bounding box dictionary containing the coordinates of the bounding box for the object is created.

C. Behaviour Control and Localization

The behaviour control and localization of robots was handled by the master node. To determine the position of a robot, the data from the images captured by the robot were used. First, the robot captures its environment with its camera and then sends this data to the master node. The master node then performs object detection using OpenCV to detect the markers present in the image. Due to the 160-degree field of view of the Raspberry Pi camera, the robot is always able to see at least one marker.

The master node uses the colour of the marker and the robot's distance from it to determine where the robot is on the field. The master node also searches for the ball using the bottom camera. If the ball is in view, then the robot tries to move in the direction of the opponent's post. Otherwise, the robot is sent a command to rotate around its axis and search for the ball again.

To make the ball move towards the post, images from the top camera were analysed. If the post is detected in the image, then the robot is sent a command to move forward a few steps, then take new images to recalibrate its path. If the post is not in the image captured, then the robot is sent commands to rotate around the ball until the post is in view.

To have higher chances of scoring, both teammates were allowed to go after the ball so that even if one robot lost the ball, the other would try to repossess it. When the ball was lost (out of reach of the robot or collected by the opponent), both teammates would also go towards it and try to get it back.

D. Communication

To make the robots communicate, ROS is installed on the Raspberry Pi 3 computer as well as on a master computer (a laptop). Each robot served as a node on the ROS network and sent image data to the master node at intervals. The master node used the received information to detect the objects around the robot and then plot a course of action for each robot.

IV. RESULTS AND DISCUSSION

This section presents pictorial results of the robots during the testing stages of implementation and a few snapshots from the actual Robosoccer competition. The software implementation of the results can be found at the github repository (Repository, 2022).

A. Robot Movement

Figure 8 shows a snapshot of the robot movement on the field in the testing environment. The robot movement is guided by the detection of the ball. In the figure the robot closest to the ball (on the left) has detected the ball and is moving towards it while the robot on their far end of the pitch is searching for the ball.

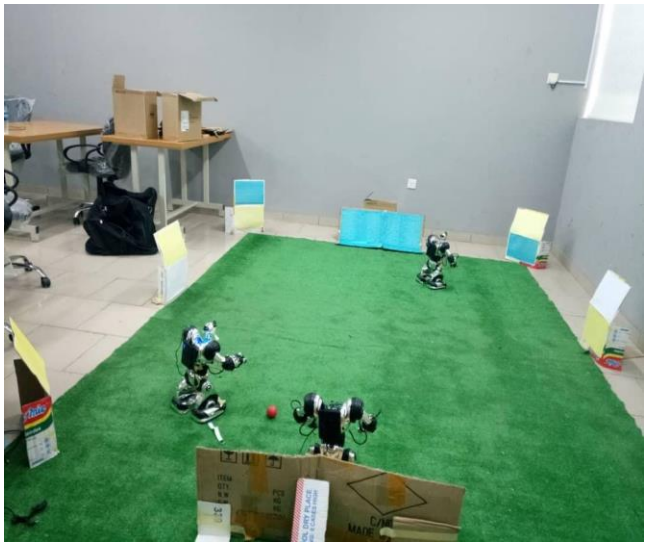

Figure 8: Snapshot of the robots moving on the field.

B. Robot Ball Passing

The task of passing the ball was quite challenging for the robots because of the anatomy of the Robosapeian robots. However, it was possible for one robot to knock the ball in the direction of the other robot as illustrated in Figure 9. Sometimes the ball is knocked way out of the path of the receiving robot as shown in Figure 8. In such a scenario the

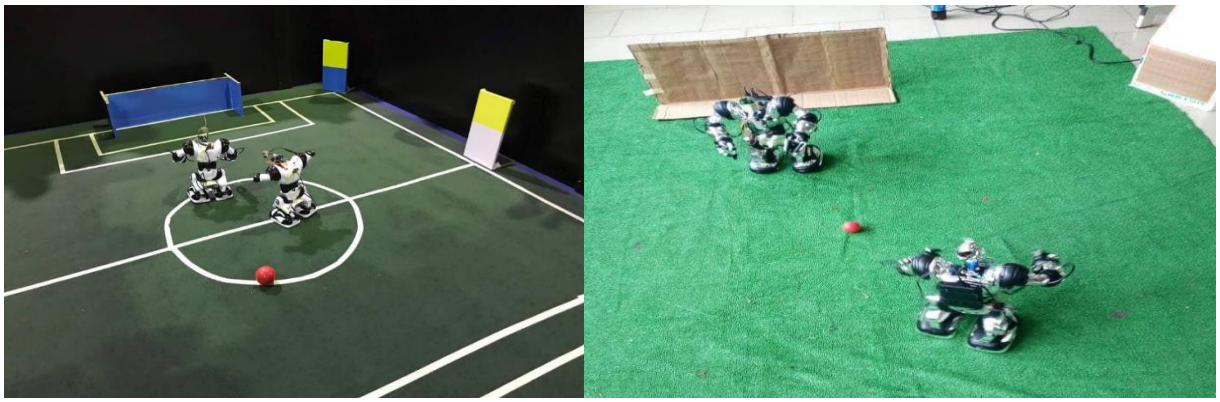

Figure 9: The robots can pass to one another by knocking the ball

receiving robot will have to resume searching for the ball and try to take possession of the ball once more.

C. Robot Scoring

In the competition as is that case in real-life soccer, a goal is scored when the ball is out over the goal line by the opponent. The competition also had the penalty option where the robot must move the ball over the goal line or successfully knock the ball over the goal line. Most matches were decided on penalties. Figure 10 shows one of the robots in the process of taking a penalty.

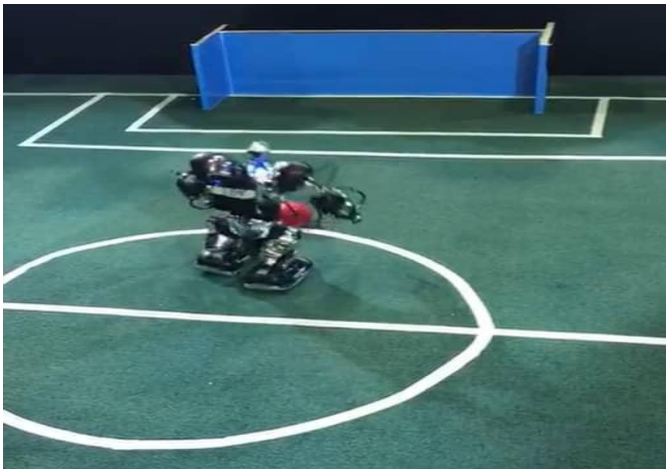

Figure 10: The robot in the process of taking a penalty (in the competition).

This section presents pictorial snapshots of the robots during the testing stages of our implementation and a few snapshots from the actual Robosoccer competition.

D. Object Detection Performance

So far in the previous subsections A through C, a pictorial representation of our results, Table 1 presents a quantitative analysis of results pertaining to the object detection task. In Table 1, the average precision and average recall are calculated using the COCO evaluation metrics (Lin *et al*, 2014). The results show that at a max detection of 100, the average precision is above 84% across different Intersection of Union(s) (IOUs). This indicates that the model can accurately localize and detect the correct object in an image on average 84 times out of 100 detections. The results for the average recall tell a similar story describing how accurate localization and detection across the object classes vary at different max-detections. In summary both the average precision and recall indicate a very good performance for the object detection model.

Table 1: A performance evaluation (COCO) of the YOLOv2 object detection showing the average precision and average recall

Metric	IOU	Area	Max-detection	Value
Average Precision	0.50:0.95	All	100	0.85
Average Precision	0.50	All	100	0.80
Average Precision	0.75	All	100	0.88
Average Recall	0.50:0.95	All	1	0.79
Average Recall	0.50:0.95	All	10	0.85
Average Recall	0.50:0.95	All	100	0.89

V. CONCLUSION

This work presents a holistic overview of the various mechanisms implemented for the “ICog Labs Robosoccer competition”. To achieve robot coordination, an object detection driven mechanism is adopted that aims to first enable the robot to detect the ball then move in the direction of the ball. Thus, the robot is either in a ball searching phase where it is trying to detect the ball or a goal searching phase where after detecting the ball it attempts to move the ball to the goal post for a goal.

The object detection modules guiding the behaviour of the robot rely on the efficiency of the YOLOv2 object detection model for detecting objects on the field such as the ball and goal post. Other challenges lay with the robot's anatomy where control of the ball was difficult because of the robot's curved feet, however due to the rule constraints of the Robosoccer competition, very little alterations could be done to the specified robots used in the work. Future work would focus on implementing an efficient tracking mechanism and a path planning mechanism. Regarding the competition, the current implementation was however relatively successful as the team was able to finish in the top 3 in the competition.

REFERENCES

iCog Labs (2018). iCog Makers Robosoccer cup Rules and Regulations. Available online at: https://icog-labs.com/afrobocup_rules_regulations_2018.pdf. Accessed on January 3, 2021.

Behnke, S., J. Müller and M. Schreiber. (2005). Toni: A soccer playing humanoid robot. In Robot Soccer World Cup, 59-70 Springer: Heidelberg, Berlin.

Bochkovskiy, A.; C. Y. Wang and H. Y. M. Liao. (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934, Available online at: https://arxiv.org/abs/2004.10934. Accessed on November 13, 2021.

Ferrein, A. and Steinbauer, G. (2016). 20 Years of RoboCup. Künstl Intell, 30: 225–232.

Jin, X.; Z. Yang; H. Wang; J. Yin; C. Wang and F. Wu. (2020). Improvements of SYCU Humanoid Robot. In Journal of Physics, Conference Series 1693(1): 012204.

LeCun, Y.; Y. Bengio and G. Hinton. (2015). Deep learning. Nature. May; 521 (7553): 436-444.

LeCun, Y.; D. Touresky, G. Hinton and T. Sejnowski. (1988). A theoretical framework for back-propagation. Proceedings of the 1988 connectionist models summer school, CMU, Pittsburgh, Pa: Morgan Kaufmann, (1): 21-28.

Lindsay, G. W. (2021). Convolutional neural networks as a model of the visual system: Past, present, and future. Journal of cognitive neuroscience, 33(10), 2017-2031.

Lin, T.Y; M. Maire; S. Belongie; J. Hays; P. Perona; D. Ramanan; P. Dollár and C. L. Zitnick. (2014) Microsoft coco: Common objects in context. Presented at the European conference on computer vision, 740-755, Springer, Cham.

Redmon, J., and Farhadi A. (2017). YOLO9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition, 7263-7271 IEEE:USA.

Redmon, J.; S. Divvala; R. Girshick and A. Farhadi. (2016). You only look once: Unified, real-time object detection. Proceedings of the IEEE conference on computer vision and pattern recognition, 779-788 IEEE:USA.

Repository (2022) Robosapien Object Detection using Darkflow. Available online at https://github.com/ogbanugot/Robosapien-Object-Detector-using-Darkflow. Accessed on February 15, 2022.

Schmidt, R. M.; F. Schneider and P. Hennig. (2021). Descending through a crowded valley-benchmarking deep learning optimizers. Paper presented at the International Conference on Machine Learning 9367-9376 PMLR:USA.

Spensieri, D; E. Åblad; R. Bohlin; J.S. Carlson and R. Söderberg (2021). Modeling and optimization of implementation aspects in industrial robot coordination. Robotics and Computer-Integrated Manufacturing, (69):102097.

Wang, D; H. Deng and Z. Pan (2020). Mrcdrl: Multi-robot coordination with deep reinforcement learning. Neurocomputing, (406): 68-76.

Zhang, A.; Z. C. Lipton; M. Li and A. J. Smola. (2021). Dive into deep learning. arXiv preprint arXiv:2106.11342. Available online at: https://arxiv.org/abs/2106.11342, Accessed on November 13, 2021.

Zhou, D. X. (2020). Theory of deep convolutional neural networks: Downsampling. Neural Networks, (124): 319-327.