A Simulation System Based on ROS and Gazebo for RoboCup Middle Size League

Weijia Yao, Wei Dai, Junhao Xiao, Huimin Lu, Zhiqiang Zheng

***Abstract*—Originated from Robot World Cup Middle Size League (RoboCup MSL), this paper discusses the design and implementation of a robot soccer simulation system based on ROS and Gazebo. Its aim is to test multi-robot collaboration algorithms. After building the Gazebo models, a model plugin is written to realize the robot's basic motions including omnidirectional locomotion, ball-dribbling and ball-kicking. To integrate the model plugin with the real robot code, ROS nodes related to low-level controllers are replaced with the model plugin. In addition, nodes related to behavior control and global information processing are modified to corresponding model plugins as well. Finally, multiple robot models are spawned into a simulation world and collaborate with others. We have already used the simulation system to successfully test and debug the multi-robot collaboration algorithms for MSL. Furthermore, after minor modifications, it can also be applied to the research of other multi-robot systems. Therefore, the simulation system is sufficient in simulating multi-robot collaboration.**

***Index Terms*—simulation system, robot soccer, multi-robot, Gazebo, ROS, RoboCup.**

I. Introduction

RoboCup [1] is an annual international competition and academic activity for the development of artificial intelligence, intelligent robotics and related fields. Currently it has five major competition domains, each with several leagues. Among them, RoboCup Soccer Middle Size League (MSL) [2] is the closest to real soccer game in terms of competition rules and intensity. One of the fundamental research issues of MSL is multi-robot collaboration [3, 4].

Since testing multi-robot collaboration algorithms on real robots is costly and difficult, an appropriate simulation system is necessary. In fact, there are many simulators either commercially available or open source. One of the typical simulators is Webots [5]. It is a mobile robotics simulation software that features complex robotic setups, customization of robot properties and large user base. But it is not open source. Soccer Server [7], in the beginning, is the official simulator of RoboCup. It is later improved and renamed as rcsoccersim. Another simulator, Übersim [6], developed by Carnegie Mellon University, is designed as a robot development tool for the RoboCup Small Size League. It provides a high-fidelity simulation environment and reconfigurable robot classes. But the robot models can only be parameterized at compile time. Some teams, such as WinKit [8] from Kanazawa Institute of technology, develop their own simulation systems for their robots. But most of these simulators are only applicable to specific robots.

Apart from two-dimensional simulators, a generic three-dimensional physical simulation system named Spark [9] is adopted by RoboCup Simulation League. Another simulation system used in RoboCup is SimRobot [10], which supports rigid body dynamics and a variety of sensors and actuators. However, these two simulators are not widely used in real games like MSL.

In this paper, the open source simulator Gazebo [11] is adopted to simulate the motions of a soccer robot. The main reason we use Gazebo as the simulator is that Gazebo offers a convenient interface with Robot Operating System (ROS) [12], which is used in our real robot code. In addition, Gazebo also features 3D simulation, multiple physics engines, high fidelity models, huge user base and etc. Therefore, the simulation system based on ROS and Gazebo can take advantage of many state-of-the-art robotics algorithms and useful debugging tools built in ROS. It can also benefit from or contribute to the active development communities of ROS and Gazebo in terms of code reuse and project co-development.

The remainder of this paper is organized as follows. Section II introduces the creation of simulation models and a simulation world. Section III presents the realization of a single robot's basic motions by a Gazebo model plugin. Furthermore, in section IV, the model plugin is integrated with the real robot code so that several robot models are able to reproduce real robots' behavior. Finally, in section V, three tests are conducted to validate the effectiveness of the simulation system. Section VI concludes the paper and summarizes the future work.

II. Simulation Models and a Simulation World

Gazebo models, which consist of links, joints(optional), plugins(optional) and etc., are specified by SDF(Simulator Description Format) [13] files whereas a simulation world, which determines lighting, simulation step size, simulation frequency and other simulation properties, is specified by a world file.

A. Simulation models

Models used in this simulation system include the robot (NuBot [14]) model, the soccer field model and the soccer ball model.

1) Robot model: It is composed of a chassis link without any joint. TABLE I lists some important properties specified in the robot model SDF file. Besides, another two important

Weijia Yao, Wei Dai, Junhao Xiao, Huimin Lu and Zhiqiang Zheng are with College of Mechatronics Engineering and Automation, National University of Defense Technology, China (e-mail: weijia.yao.nudt@gmail.com, 975475085@qq.com, junhao.xiao@ieee.org, lhmnew@nudt.edu.cn, zqzheng@nudt.edu.cn). Weijia Yao is the corresponding author.

properties, mesh and collision that are used for visualization and collision detection respectively, are illustrated in Fig. 1. They are drawn by open source 3D drawing tool SketchUp [15]. Note that the collision element is not a duplicate of the model's exterior but a simplified cylinder with the same base shape and height as the model exterior. This simplification does not affect the simulation effect but improves the simulation speed. For the same purpose, we do not model the real robot's physical mechanisms, such as omnidirectional locomotion, ball-dribbling, ball-kicking and omni-vision camera mechanisms, and therefore there is no need for any joint in this model. The simplification can be justified by the simulation system's design purpose: to test multi-robot collaboration strategies or algorithms. Therefore the emphasis of the simulation system is on the final effect of robots' basic motions but not the complicated physical processes involved. The physical mechanisms' capabilities can be realized by a Gazebo model plugin that will be discussed in Section III.

TABLE I.
PROPERTIES OF THE ROBOT MODEL

Property name	**Value**
Mass [11]	31 kg
Moment of inertia [11]	I_{zz} = 2.86 kg•m^2 , I_{xx}= I_{yy}= I_{xy}= I_{xz}= I_{yz}=0
Friction coefficient	0.1
Velocity decay	Linear: 0. Angular: 0
Model plugin	nubot_gazebo

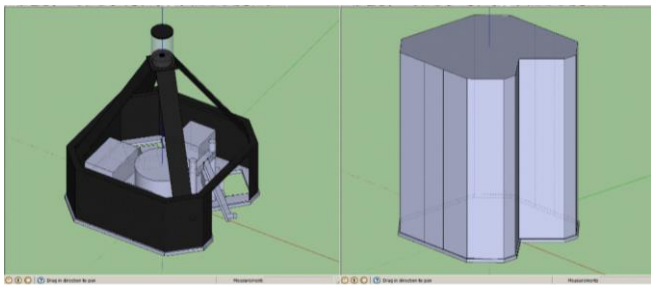

Fig. 1. *Mesh* and *collision* properties of the robot model. (a) *Mesh* property; (b) *collision* property.

2) Soccer field model: According to 2015 RoboCup MSL competition rules [17], we use SDF file, OGRE [18] material scripts and images of the goal net, field ground and field lines to build the soccer field model. The collision elements are composed of each parts' corresponding geometry.

3) Soccer ball model: The soccer ball model is built with the same attributes of a defined FIFA standard size 5 soccer ball that is played in RoboCup MSL. The pressure inside the model is neglected and the collision element is a sphere of the same size of the soccer ball.

TABLE II.
PROPERTIES OF THE SIMULATION WORLD

Property name	**Value**
Physics engine	Open Dynamics Engine (ODE) [14]
Max step size	0.007 s
Real time update rate	150 Hz
Gravity	-9.8 m/s^2
Models to spawn	ground_plane, soccer field, left_goal, right_goal

B. The Simulation world

The world file specifies the simulation background, lighting, camera pose, physics engines, simulation step size and etc. Some important properties of the simulation world are listed in TABLE II. Finally, a simulation world with three robots and a soccer ball is created (see Fig. 2).

Fig. 2. The simulation world

III. A SINGLE ROBOT'S BASIC MOTIONS REALIZATION

To realize a single robot's basic motions, a Gazebo model plugin named "nubot_gazebo" is written. A model plugin is a shared library that attached to a specific model and inserted into the simulation. It can obtain and modify the states of all the models in a simulation world. We will discuss the "nubot_gazebo" model plugin in three parts.

A. Overview of the "nubot_gazebo" plugin

When "nubot_gazebo" plugin is loaded at the beginning of a simulation process, its tasks include:

- Obtaining parameters of the soccer ball model's name, ball-dribbling distance threshold, ball-dribbling angle threshold and etc. from ROS parameter server.
- Setting up ROS publishers, subscribers, service servers and a dynamic reconfigure server.
- Binding model plugin update function that runs in every simulation iteration.

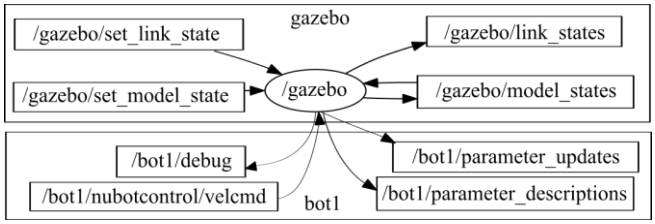

Fig. 3. The computation graph of the model plugin

The model plugin starts running automatically when a robot model is spawned. For example, when the robot model "bot1" is spawned, a computation graph is created and visualized by ROS tool *rqt_graph* (see Fig. 3). In Fig. 3, there is only one node called "/gazebo", which publishes (represented by an arrow pointing outward) and subscribes (represented by an arrow pointing inward) several topics enclosed by small rectangles. The topics inside the "gazebo" namespace are created by a ROS package called *gazebo_ros_pkgs*, which provides wrappers around the stand-alone Gazebo and thus enables Gazebo to make full use of ROS messages, services and dynamic reconfigure. Those inside the "bot1" namespace are created by the model plugin. All the topic names are self-explanatory. For instance,

messages on */bot1/nubotcontrol/velcmd* topic are used to control the robot model's velocity.

Although the physical mechanism of the omni-vision camera is not simulated, the robot model is still able to obtain information of other models' positions and velocities by subscribing to the topic */gazebo/model_states*. In addition, ball-dribbling and ball-kicking are realized by calling corresponding ROS services (not illustrated in Fig.3). They will be discussed in the following part.

B. Motions realization

A single robot's basic motions include omnidirectional locomotion, ball-dribbling and ball-kicking. For better explanation, two reference frames are created: the world frame denoted by the subscript *w* and the robot body frame denoted by the subscript *r* as shown in Fig. 4. The x_r- axis is in the ball-kicking direction. Its corresponding unit vector is denoted by $\overrightarrow{D_K}$, of which the representation in the robot body frame is $D_{Kr} = [1\,0\,0]^T$ and in the world frame D_{Kw}:

$$D_{Kw} = R_{wr} D_{Kr} \tag{1}$$

where R_{wr} is the rotation matrix representing the orientation of the robot body frame *r* relative to the world frame *w*. $\overrightarrow{D_T}$ is the unit vector pointing from the geometry center of the robot to that of the soccer ball.

1) Omnidirectional locomotion: Gazebo's built-in functions *SetLinearVel* and *SetAngularVel* are used to make the robot model move in any direction given any translation vector and rotation vector respectively.

2) Ball-dribbling: If the distance between the robot and the soccer ball is within a distance threshold and the angle between $\overrightarrow{D_K}$ and $\overrightarrow{D_T}$ is also within an angle threshold, then the dribble condition is satisfied and the robot is able to dribble the ball. Under this condition, to realize ball-dribbling, the soccer ball's pose is directly and continuously set by Gazebo's built-in function to continually satisfy the dribble condition. In other words, the vector pointing from the geometry center of the robot to that of the soccer ball ($\overrightarrow{P_{rs}}$) is set as follows:

$$\overrightarrow{P_{rs}} = gap * \overrightarrow{D_K} \tag{2}$$

where gap is the specified distance (smaller than the dribble distance threshold) between the robot and the soccer ball.

3) Ball-kicking: Similarly, ball-kicking is realized by giving the soccer ball a specific velocity at the start of the kicking process. There are two ways of kicking: the ground pass and the lob shot. In the ground pass, the soccer ball does not lose contact with the ground so its initial velocity vector is calculated in the x_w-y_w plane:

$$\overrightarrow{v_{gp}} = v_0 \overrightarrow{D_K} \tag{3}$$

where v_0 is the desired initial speed. As for the lob shot, the soccer ball is kicked into the air so its speed in the z-direction should also be taken into account. Since the air resistance is trivial compared with the gravity effect, it is reasonable to assume that the ball's flight path is a parabola. For convenience, a planar frame X-Y of which the origin is the center point of the soccer ball is set up. Its X- and Y-axes are parallel to x_r- and z_r- axes respectively as shown in Fig. 4.

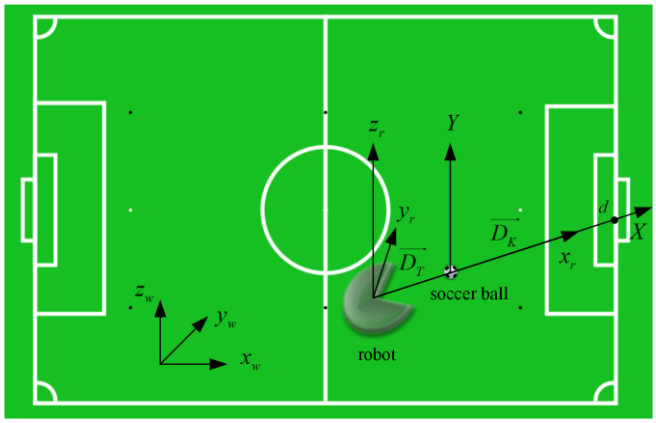

Fig. 4. Reference frames and notations for motions realization

The following calculation is discussed in this planar X-Y frame. The soccer ball is regarded as a particle so the equation of its flight curve is defined by:

$$y = f(x) = ax^2 + bx + c \tag{4}$$

where *a*, *b* and *c* are parameters to be solved. The distance between the origin and the crosspoint of the X-axis and the common perpendicular of the X-axis and the goal line is denoted by *d*; the goal height is *H* (measured from the ground to the lower surface of the horizontal goal pole); the radius of the ball is *r*. Let *h=H-2r*, so when the ball flies through the point *(d, h)* this shot is successful. In other words,

$$\begin{cases} f(x=0) = 0 \\ f(x=d) = h \end{cases} \tag{5}$$

Suppose the soccer ball's velocity along the X-axis v_X is constant during the flight, so the displacement along X-axis is calculated by $x(t) = v_X t$ and substituted in (4) we obtain:

$$y(t) = a{v_X}^2 t^2 + b v_X t + c \tag{6}$$

$$y''(t) = 2a{v_X}^2 = -g \tag{7}$$

where g is the gravitational constant. Combining with (4), (5) and (7) we obtain equation (8).

$$\begin{cases} a = -\dfrac{g}{2{v_X}^2} \\ b = \dfrac{h}{d} + \dfrac{g}{2{v_X}^2} d \\ c = 0 \end{cases} \tag{8}$$

Therefore the soccer ball's initial velocity in Y-axis is:

$$v_Y = y'(t)|_{t=0} = b v_X \tag{9}$$

Above all, the initial velocity in the world frame is:

$$v_{ls,w} = (v_X \cdot D_{Kw,x}, v_X \cdot D_{Kw,y}, b v_X) \tag{10}$$

where $D_{Kw,x}$ and $D_{Kw,y}$ are the x- and y- component of D_{Kw} respectively. In fact, *h* is given a smaller value in the simulation to guarantee that the ball will always be shot into the goal. Note that v_X has an upper limit given by:

$$-\frac{b}{2a} < d \tag{11}$$

Combining with (8) and (11) we get this upper limit denoted by v_{thres}:

$$v_X < v_{thres} = d\sqrt{\frac{g}{2h}} \tag{12}$$

C. Single robot motions test

To test single robot's basic motions, four behavior states are defined as follows: CHASE_BALL, DRIBBLE_BALL (including two sub-states MOVE_BALL and ROTATE_BALL), KICK_BALL, RESET and HOME. The robot model performs these motions following the behavior states transfer graph as shown in Fig. 5. The test results (see Fig. 6) prove that the "nubot_gazebo" model plugin realizes basic motions successfully.

IV. Model Plugin and Real Robot Code Integration

The "nubot_gazebo" model plugin discussed above has realized a single robot's basic motions without the need to model the real robot's physical mechanisms. However, in order to test multi-robot collaboration algorithms, it is significant to integrate the model plugin with the real robot code.

In the real robot code, there are seven nodes in total (see Fig. 7). The five nodes enclosed by shadowed ellipses are directly related to the robot's hardware. The other two nodes, "world_model" node and "nubot_control" node, process global information and control robot behavior respectively. In addition, there is a Windows program dubbed Coach that receives and visualizes information from each robot and sends basic commands such as game-start, game-stop, kick-off, corner-ball and etc. via UDP protocol to them in return.

To integrate the real robot code with the model plugin, those five nodes related directly to hardware should be replaced with the model plugin. This successful replacement requires an appropriate interface, in other words, correct ROS messages-passing and services-calling between them. Finally, the data flow of the integration of the real robot code and the model plugin is shown in Fig. 8. There are three noticeable changes described as follows:

1) "nubot_control" and "world_model" nodes converted to model plugins: As a result, the namespace has been changed from "nubot" to "gazebo". There are two reasons why "nubot_control" and "world_model" nodes are converted to model plugins.

The first reason is for model completeness. Because only if all of the three model plugins are attached to a robot model can the model simulate the complete behavior of a corresponding real robot. Therefore, these two nodes should be converted to model plugins so that they will be loaded automatically once the robot models are spawned.

The second reason is for name-prefixing. Because all the robot models use the same model plugins and are created into one simulation world, they cannot distinguish their own messages and services from others'. In this case, it is necessary for each models' name to be used as a prefix to their own topic names or service names. Therefore, the simulation robots can subscribe to their own topics or response to their own services. These prefixes-the model names-are obtained by using model plugins.

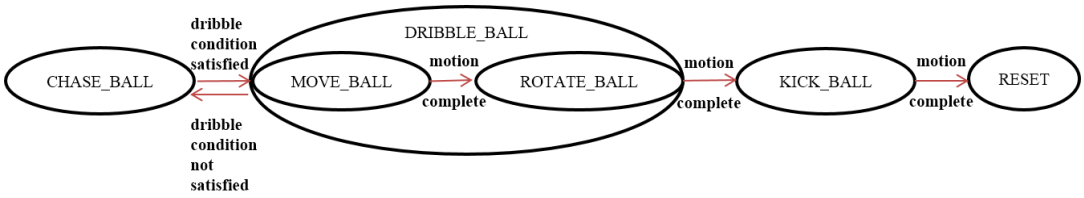

Fig. 5. Single robot behavior states transfer graph

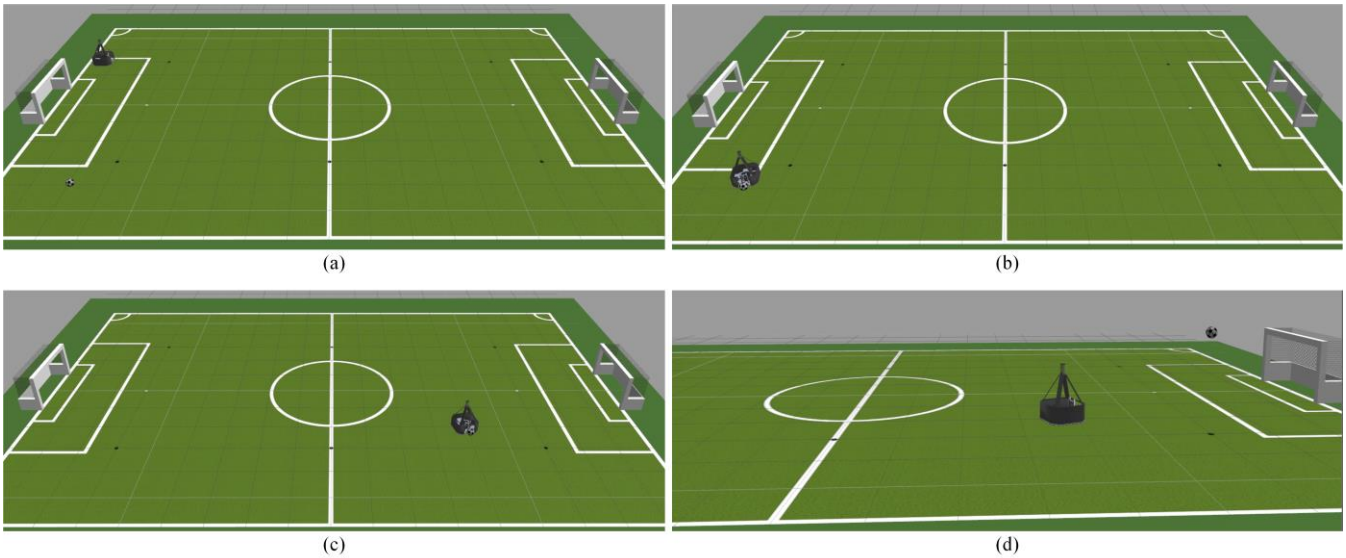

Fig. 6. Single robot simulation result. (a) Initial state; (b) CHASE_BALL state; (c) DRIBBLE_BALL state; (d) KICK_BALL state;

In addition, the conversion from these nodes to model plugins is not complicated; the essential method is to inherit from "ModelPlugin" class and overload the "Load" member function [21]. However, if "world_model" and "nubot_control" are not converted to model plugins, an extra mechanism, such as a well-written bash script, is necessary to guarantee that they correspond to the appropriate robot models.

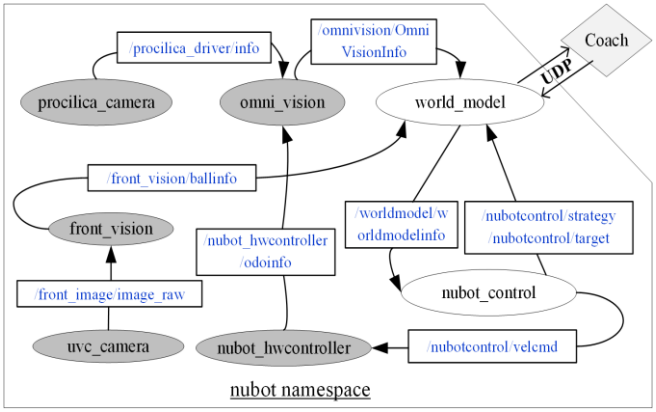

Fig. 7. The data flow graph of the real robot code.

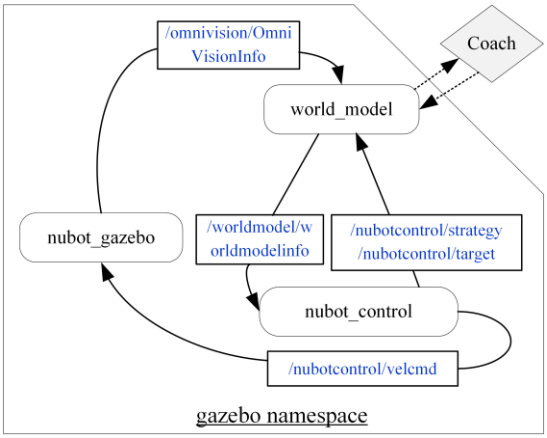

Fig. 8. The data flow graph of the integration of the real robot code and the model plugin.

2) An intermediary node used for communication between robots and the Coach: As can be seen in Fig.7 and Fig.8, the connection between "world_model" and the Coach has been changed from two full lines to two broken lines, indicating that the communication is indirect in the latter situation. In fact, the real robot code uses blocking socket network communication with the Coach. Thus if several robots are spawned in a simulation world, only one robot is able to receive information from the Coach. The solution to the problem is either using non-blocking socket programming or using an intermediary node as a bridge. We adopt the latter method to create a node called "/coachinfo_publisher" to successfully solve this problem. This node communicates via UDP protocol with the Coach and then publishes messages on a specific topic so that each robot can subscribe to it.

3) Gaussian noise: Gaussian noise is added to the position and velocity information obtained by the robot model to mimic the real world situation.

Finally, we spawn two robot models "bot1" and "bot2" into a simulation world and obtain the computation graph as shown in Fig. 9. Note that all the model plugins are contained in the "/gazebo" node and the topic names are all prefixed by corresponding model names.

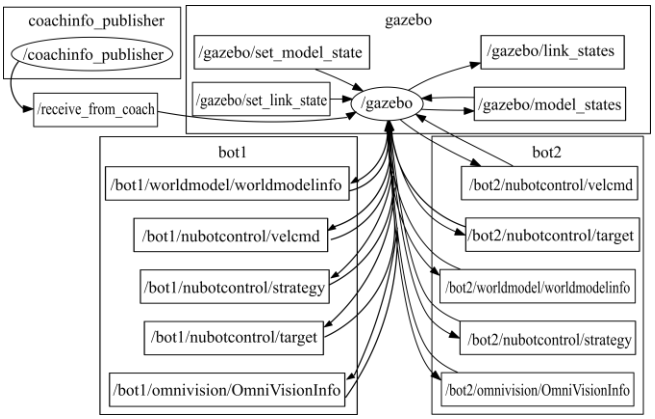

Fig. 9. The computation graph of the simulation with two robot models

TABLE III.
CONFIGURATION OF TWO COMPUTERS

	Operating system and software	IP address	Subnet mask
Computer1	Ubuntu 14.04 64 bits; ROS Indigo; Gazebo 5.0	172.16.53.203	255.255.0.0
Computer2	Windows 7 64 bits; Visual Studio 2013	172.16.53.220	255.255.0.0

V. MULTI-ROBOT SIMULATION TEST

To test multi-robot simulation, we need two computers, Computer 1 and Computer 2, connected by a network cable. Computer 1 is used to run Gazebo and simulation models whereas Computer 2 is used to run the Coach program. Their configurations are listed in TABLE III.

Firstly, we test the communication between the model plugins and the Coach program. In this test, three robot models and three static obstacles (static rival robots) are spawned into the simulation world. The test result shows that the positions of the models are displayed correctly in the Coach window. And in reverse, the robot models are able to receive commands from the Coach program. So the communication is a success.

Secondly, given static obstacles the same as the first test, we test whether the robot models can make motions corresponding to the real robot code. The result proves that the robot models can perform motions as expected.

Lastly, by attaching separate model plugins to the rival robot models and reversing the reference frames in the "/nubot_gazebo" plugin, two sides of robot models are able to compete against each other as shown in Fig. 10.

To sum up, these tests prove that the interface between the model plugins and the real robot code is successfully realized. And the simulation system is able to simulate both sides of robots' motions.

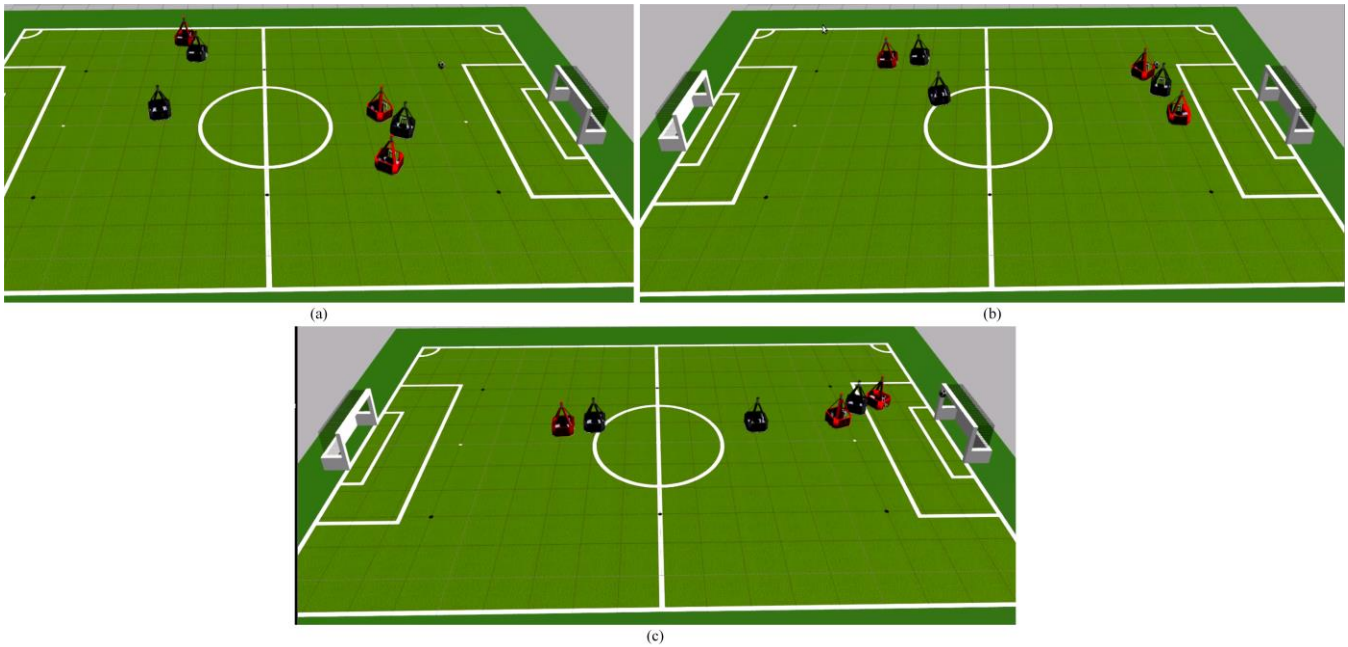

Fig. 10. Two sides of robot models compete in a soccer game. (a) The initial state of the robot models (black) and the rival robot models (red); (b) three robots on the right side of the soccer field are chasing the soccer ball; (c) Black robots shoot the goal.

VI. CONCLUSION AND FUTURE WORK

A. Conclusion

In order to test multi-robot collaboration algorithms of real robot code, we set up a robot soccer simulation system using ROS and Gazebo model plugins. This process comprises of three stages. The first stage is constructing Gazebo models with visualization, collision elements and other properties. The second stage is writing a Gazebo model plugin to realize the basic motions such as omnidirectional locomotion, ball-dribbling and ball-kicking. The last stage is to integrate the model plugin with the real robot code. Finally, the test results manifest that the simulation system is able to simulate multi-robot collaboration.

Although the simulation system uses NuBot as the robot model, the design and implementation methods can be widely applied to other robots as well. Specifically, this simulation system can be easily adapted for other soccer robots by modifying corresponding messages and services. Overall, the simulation system enriches and develops the research of multi-robot simulation.

B. Future work

Further study will focus on the modeling of omnidirectional wheels and improvement of the simulation speed.

ACKNOWLEDGMENT

Our works are supported by National Science Foundation of China (No.61403409 and No. 61503401) and graduate school of National University of Defense Technology.

REFERENCES

[1] http://www.robocup2015.org/
[2] Kitano et al., "RoboCup: The Robot World Cup Initiative." in *Proc. of the 1st Int. Conf. on Autonomous agents.*, pp. 340-347, *ACM*, 1997
[3] Soetens, Robi et al., "RoboCup MSL-History, Accomplishments, Current Status and Challenges Ahead." In *RoboCup 2014: Robot World Cup XVIII*, pp. 624-635, Springer International Publishing, 2015.
[4] Candea et al., "Coordination in multi-agent RoboCup teams." in *Robotics and Autonomous Systems* 36, no. 2 (2001): 67-86
[5] https://www.cyberbotics.com/overview
[6] Browning et al., "Übersim: a multi-robot simulator for robot soccer." In *Proc. of the 2nd Int. joint Conf. on Autonomous agents and multiagent systems*, pp. 948-949. ACM, 2003
[7] Itsuki, Noda. "Soccer server: a simulator of RoboCup.". In *JSAI AI-Symposium 95: Special Session on RoboCup*. 1995
[8] http://demura.net/9ode/442.html
[9] Obst, Oliver, and Markus Rollmann. "Spark–a generic simulator for physical multi-agent simulations." in *Multiagent System Technologies*, pp. 243-257. Springer Berlin Heidelberg, 2004.
[10] Laue et al., "SimRobot–a general physical robot simulator and its application in robocup." in *RoboCup 2005: Robot Soccer World Cup IX*, pp. 173-183. Springer Berlin Heidelberg, 2006
[11] Koenig Nathan, and Andrew Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator." in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference*, vol. 3, pp. 2149-2154. IEEE, 2004
[12] Quigley et al., "ROS: an open-source Robot Operating System." in *ICRA workshop on open source software*. 2009. vol. 3, no. 3.2, p. 5
[13] http://sdformat.org/
[14] Yu, Wentao et al., "NuBot team description paper 2010." *Proc. RoboCup 2010 Singapore*, CD-ROM (2010).
[15] http://www.sketchup.com/
[16] Cui Qingzhu, "The Modeling and Control Based on the Dynamics for the Omni-directional Mobile Robot," M.S. thesis, School of Gratuate, Nat. Univ. of Def. Tech., Changsha, China, 2012
[17] MSL Technical Committee, "Middle Size Robot League Rules and Regulations for 2015 Version - 17.2 20141231, 5", 2015-5-20
[18] http://www.ogre3d.org/
[19] http://opende.sourceforge.net/.
[20] http://wiki.ros.org/gazebo_ros_pkgs
[21] http://gazebosim.org/tutorials?tut=ros_plugins&cat=connect_ros