Building Software System and Simulation Environment for RoboCup MSL Soccer Robots Based on ROS and Gazebo

Junhao Xiao, Dan Xiong, Weijia Yao, Qinghua Yu, Huimin Lu and Zhiqiang Zheng

Abstract This chapter presents the lesson learned during constructing the software system and simulation environment for our RoboCup Middle Size League (MSL) robots. The software is built based on ROS, thus the advantages of ROS such as modularity, portability and expansibility are inherited. The tools provided by ROS, such as RVIZ, rosbag, rqt_graph just to name a few, can improve the efficiency of development. Furthermore, the standard communication mechanism (topic and service) and software organization method (package and meta-package) introduces the opportunity for sharing codes among the RoboCup MSL community, which is a fundamental issue to forming hybrid teams. As known, to evaluate new algorithms for multi-robot collaboration on real robots is expensive, which can be done in a proper simulation environment. Particularly, it would be nice if the ROS based software can also be applied to control the simulated robots. As a result, the open source simulator Gazebo is selected, which offers a convenient interface with ROS. In this case, a Gazebo based simulation environment is constructed to visualize the robots and simulate their motions. Furthermore, the simulation has also been used to evaluate new multi-robot collaboration algorithms for our NuBot RoboCup MSL robot team.

Keywords Robot soccer · Gazebo · ROS · Multi-robot collaboration · Simulation

J. Xiao · D. Xiong · W. Yao · Q. Yu · H. Lu (✉) · Z. Zheng
College of Mechatronics and Automation, National University of Defense Technology, Changsha 410073, China
e-mail: lhmnew@nudt.edu.cn

J. Xiao
e-mail: junhao.xiao@ieee.org

D. Xiong
e-mail: xiongdan@nudt.edu.cn

W. Yao
e-mail: weijia.yao.nudt@gmail.com

Q. Yu
e-mail: yuqinghua163@163.com

Z. Zheng
e-mail: zqzheng@nudt.edu.cn

A. Koubaa (ed.), *Robot Operating System (ROS)*, Studies in Computational Intelligence 707, DOI 10.1007/978-3-319-54927-9_18

1 Introduction

RoboCup,[1] short for Robot World Cup is an international initiative to foster the research in artificial intelligence (AI) and mobile robotics, by offering a publicly appealing, but formidable challenge. In other words, it is a perfect combination of sport and technology, thus has attracted many researchers and students. Since founded in 1997, it has promoted the research field for almost two decades [1, 2]. The final goal of RoboCup is that a team of fully autonomous humanoid soccer robots will beat the human World Cup champion team by 2050 [3].

Besides soccer games, other competition stages have been introduced into RoboCup along with its growth. As a result, the contest currently has six major competition domains, namely RoboCup Soccer, RoboCup Rescue, RoboCup@Home, RoboCup@Work, RoboCup Logistics League and RoboCupJunior, and each domain has several leagues and sub-leagues. More information can be found in the Robot World Cup book series published by Springer [4, 5].

For RoboCup middle size league (MSL), the robots can be designed freely as long as they stay below a maximum size and a maximum weight. The game is played on a carpet field at the size of 18 m $\times$ 12 m, with white lines and circles as landmarks for localization. In the competition, all the robots are completely distributed and autonomous, which means they must use their own on-board sensors to perceive the environment and make decisions. According to the rules, wireless communication is allowed to share information among a team of robots, which can help the cooperation and coordination. Therefore, RoboCup MSL is a standard and challenging real-world test bed for multi-robot control, robot vision and other relative research subjects.

As one league with the longest history among RoboCup, lots of scientific and technical progresses have been achieved in RoboCup MSL, an overview can be found in [6]. Its games are also becoming more and more fluent and fierce. For example, the robots can actively handle the ball for stepping forwards, turning and stepping backwards, can make dynamic long passes, and their velocity can reach about 5 m/s, etc. Therefore, in recent years, the RoboCup MSL final has been serving as the grand finale of RoboCup, which gives the opportunity to all audiences and participants to enjoy the game together. A typical competition scenario has been drawn in Fig. 1. However, this brings lots of difficulties for new teams to catch-up, because it is not easy and very time-consuming to design and implement a team of RoboCup MSL soccer robots from the very beginning.

Since its birth in 2010, as an open source software, ROS has attracted a huge number of robotic researchers and hobbyist, which have been serving as an active and high-productive community to boost the development of ROS. The community not only optimizes the code but also stands in the front of robotic research, i.e., implementations of state-of-the-art algorithms can be found in ROS. Therefore, ROS is becoming the *de facto* standard for robotic software. Built under ROS, the robot software components can be well and easily organized. Strictly speaking, the code

[1] http://www.robocup.org/.

Fig. 1 A typical scenario of the RoboCup MSL competition: a match between TU/e and NuBot in RoboCup 2014 João Pessoa, Brazil

implementation can achieve high modularity and re-usability. Meanwhile, lots of useful tools have been provided by ROS, which can ease data logging and sharing, and code sharing among RoboCup MSL teams.

In April 2014, a question is raised among our team, i.e., whether ROS is suitable for RoboCup MSL? As a result, we decided to drop our self-developed software framework (more than 10 years development) and build the software system for our soccer robot based on ROS. Then in July, we participated RoboCup 2014 with robots with new "soul". The competition brought two positive remarks. First, the software was more robust than ever before. Second, the work was acknowledged by other teams. In this paper, we will detail the ROS based software, and hope to provide a valuable reference for building ROS based software for distributed multi-robot systems.

In addition, to evaluate new algorithms for multi-robot collaboration using real robots is very time consuming, which demands a high-fidelity simulation environment. Particularly, it would be of efficiency if the simulated robot has the same control interface as the real robot. In fact, there are many robotic simulators either commercially available or open source, such as V-REP [7], Gazebo [8], Webots [9], LpzRobots [10], just to name a few. Detailed introduction and comparison of the state-of-the-art robotics simulators can be found at [11, 12]. Among the simulators, Gazebo offers a convenient interface, i.e., from the software interface point of view, there is no difference between controlling a real robot and its Gazebo dummy. In other words, the algorithms evaluated using the simulation environment can be applied to the real robots without change. Therefore, we choose Gazebo to construct the simulation environment.

This Chapter will cover the design and implementation of the software system, simulation environment, and their interfaces for our RoboCup MSL robots. It is based on our previous work of [13, 14]. The code has been made open source, with the ROS based software can be accessed at https://github.com/nubot-nudt/nubot_ws, while the simulator can be accessed at https://github.com/nubot-nudt/gazebo_visual. A video showing a simulated match using our multi-robot simulator can be found at https://youtu.be/rMuAZGf65AE. The remainder of the chapter consists of the following topics:

- First, the background is introduced in Sect. 2.
- Second, a brief introduction of our NuBot multi-robot system is given in Sect. 3, including the mechanical structure, the perception sensors and the electrical system.
- Third, the ROS-based software is detailed in Sect. 4.
- Fourth, the Gazebo-based simulation environment is drawn in Sect. 5, with the focus on how we designed the same control interface for the real robots and simulated robots.
- Fifth, two short tutorials are given for single robot simulation and multi-robot simulation in Sects. 6 and 7, respectively.
- A short conclusion is given in Sect. 8.

2 Background

It is not trivial to design robots for highly competitive and dynamic environments like the RoboCup MSL, thus many hardware designs and software algorithms have been proposed, see [6] for an overview. In this section, we try to give a brief introduction from which the readers can acquire more details about the achievements in RoboCup MSL, and the efforts which have been done to cut down the difficulty in developing RoboCup MSL robots. Our previous works will also be introduced in this section.

RoboCup MSL has achieved scientific results in robust design of mechanical systems, sensor-fusion, tracking, world modelling and distributed multi-robot coordination. A special issue named "Advances in intelligent robot design for the Robocup MSL" was published in 2011 [15]. In this issue, the state-of-the-art research about mechatronics and embedded robot design, vision and world modelling algorithms, team coordination and strategy was presented. The paper [6] overviews the history and the current state of the RoboCup MSL competition, which also presents a plan to further boost scientific progress and to attract new teams to the league. Surveys about team strategies, vision systems and visual perception algorithms in RoboCup MSL can be found in [16–18].

Recently, some RoboCup trustees reported the history and state-of-the-art of RoboCup soccer leagues, in which they had very positive comments on the RoboCup MSL [19], e.g., "This Middle Size League has had major achievements during the last few years. Middle Size League teams have developed software that allows amazing

forms of cooperation between robots. The passes are very accurate and some complex, cooperatively made goals are scored after passing the ball, rather than just out-dribbling an opponent and playing individually". However, this brings lots of difficulties for new teams to catch-up. As a result, the number of RoboCup MSL teams has not rise for years. How to draw more teams to participate RoboCup MSL and then make contributes to RoboCup MSL is becoming a major problem. Facing this reality, the RoboCup MSL community has made efforts to reduce the difficulty in implementing RoboCup MSL teams. For example:

- The launch of ROP (Robotic Open Platform) [20], it facilitates the release of hardware designs of robots and modules under an open hardware license. In the repository, the robots named Turtle of team Tech United have been fully released.
- Another remarkable propose is to design and implement an affordable platform for RoboCup MSL, thus providing an easier starting point for any new team, i.e., the Turtle-5k project.[2] With the support from the Tech United team, the TURTLE-5k platform is developed based on the 2012 TURTLE robots, which has won the RoboCup MSL world Champion. The Value Engineering method has been employed to find out some the most cost part, where the cost could be reduced.
- Real-time efficient communication among robots is of key important for cooperation, the CAMBADA team [21] proposed a TDMA (Time Division Multiple Address) based communication protocol, which is designed for real time data sharing in a local network. CAMBADA also implemented the communication protocol, which is named real-time database tool (RTDB) [22, 23]. Furthermore, RTDB has been made open source, and several teams are using it for communication.

Although RoboCup MSL teams have made significant achievements, there are still some open problems and challenges in constructing a RoboCup MSL robot team to play with human beings:

- The robot platform should have good performance in critical aspects such as top speed and top accelerations, and be able to handle impacts. It should be easy to assemble and maintain.
- It is necessary to improve the stability of the electrical system, and the extension of sensors should be better supported.
- The robustness of the vision system should be improved to make it work reliably in both indoor and outdoor environments with highly dynamic lighting conditions.
- The software framework should support code reusing and data sharing as much as possible.

[2] http://www.turtle5k.org/.

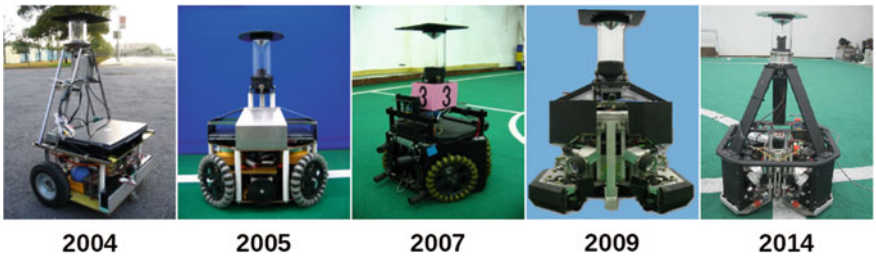

Fig. 2 The 5 generation NuBot robots

3 The NuBot Multi-robot System

Our NuBot team[3] was founded in 2004. As shown in the Fig. 2, from the very beginning, 5 generations robots have been created. It can be seen that, the NuBot robots have been always using omni-directional vision system [24, 25], and have been equipped with omni-directional chassis since the second generation [26]. We had participated in RoboCup simulation and small size league (SSL) at first. Since 2006, we have been participating RoboCup MSL actively, e.g., we have been to Bremen, Germany (2006), Atlanta, USA (2007), Suzhou, China (2008), Graz, Austria (2009), Singapore (2010), Eindhoven, Netherlands (2013), Joao Pessoa, Brazil (2014), and Hefei, China (2015) [27]. We have been also participating RoboCup ChinaOpen since it was launched in 2006. This chapter will present the software system and simulation environment for the last generation robot.

3.1 Mechanical Platform

This section draws the mechanical platform of our NuBot soccer robots. When designing the robot platform, there are several criteria to be considered. First, it should comply with the rules and regulations of RoboCup MSL, namely, its size, weight and safety concerns. Second, it should have good maneuverability in order to play against others. Lastly, because malfunctions or failures are unavoidable during the intensive and fierce RoboCup MSL games, the mechanical parts should embrace high modularity such that they are easy to assemble and maintain. To fulfil these requirements, The NuBot robots have been designed modularly as shown in Fig. 3.

Currently, the regular robot and the goalie robot are heterogeneous due to their different tasks. For the regular robot, it should be able to do the same things as a human soccer player, such as moving, dribbling, passing and shooting. Therefore, the mechanical platform is subdivided into three main modules as illustrated in Fig. 3a.

[3] http://nubot.trustie.net.

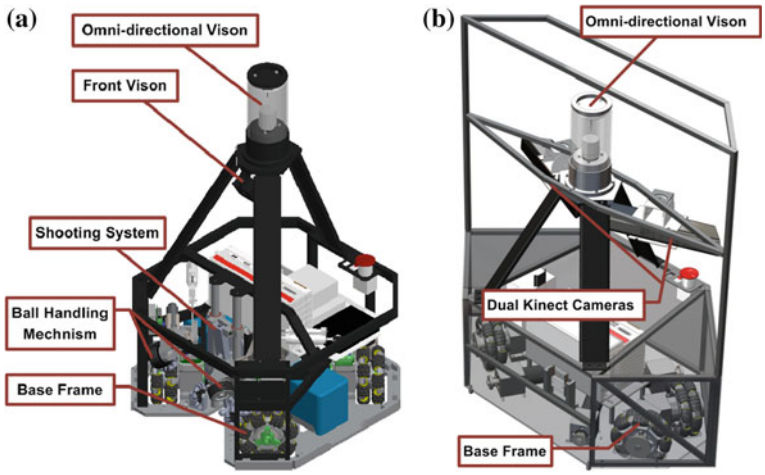

Fig. 3 **a** The regular robot. **b** The goalie robot

- the base frame;
- the ball-handling mechanism;
- the electromagnet shooting system;

For the goalie robot, the ball-handling mechanism, the electromagnet shooting device and the front vision system are removed, instead two RGB-D cameras are integrated as shown in Fig. 3b. Furthermore, to increase the side acceleration, the configuration of omni-directional wheels has also been modified.

In the below, we give a brief introduction to each part.

Base Frame The holonomic-wheeled platform, which is capable of carrying out rotation and translation simultaneously and independently, has been used by most RoboCup MSL teams [21, 28]. In the NuBot platform, our custom-designed omni-directional wheel has been utilized, as illustrated in Fig. 4a. Four such omni-directional wheels are uniformly arranged on the base as shown in Fig. 4. Despite the added costs of extra weight and extra power consumption, the four-wheel-configuration platform can generate more traction force than a normal three-wheel-configuration one, thus can improve the maneuverability. For the goalie robot, its main motion is side moving to defend coming balls. In this case, the configuration of omni-directional wheels have been modified as shown in Fig. 4c.

Ball-Handling Mechanism The ball-handling mechanism enables the robot to catch and dribble the football. As illustrated in Fig. 5, there are two symmetrical assemblies, each contains a wheel, a DC motor, a set of transmission bevel-gears, a linear displacement transducer and a support mechanism. According to the gravity and pressure from the support mechanism, the wheels are stuck to the ball when the ball is in. Therefore, they can generate various friction forces to the ball, and make it

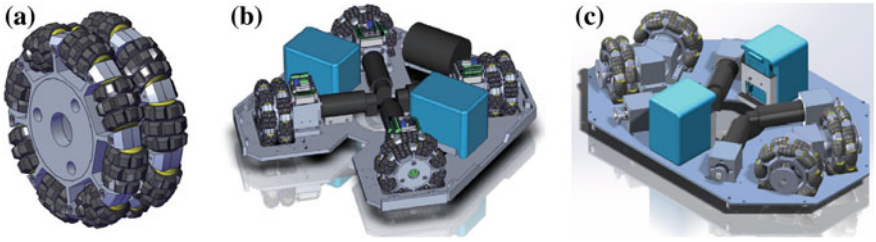

Fig. 4 **a** The custom designed omni-directional wheel. **b** Base for regular robots. **c** Base for the goalie robot

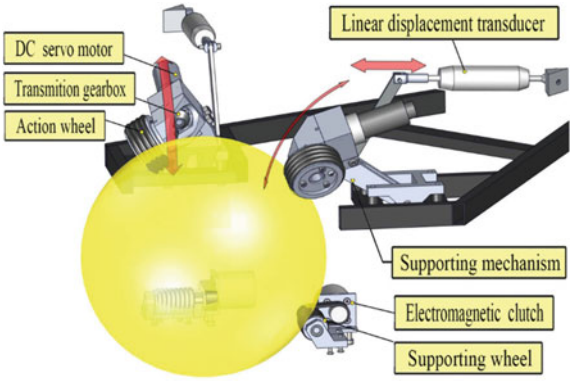

Fig. 5 The ball-handing mechanism of the NuBot

rotate in desired directions and speeds together with the soccer robot. During dribbling, the robot constantly adjusts the speed of the wheels, in order to maintain a proper distance between the ball and the robot using a closed-loop control system. This control system takes the distance between the ball and the robot as the feedback signal, which is measured by the linear displacement transducers. As the ball moves closer to the robot, the supporting mechanism will raise, then compress the transducer, otherwise the support mechanism will fall and stretch the transducer. This system effectively solves the ball-handling control problem.

Electromagnet Shooting System The shooting system enables the robot to pass and score, currently there are three ways to construct a shooting actuator, i.e., spring mechanisms, pneumatic systems and solenoids electromagnet, an overview can be seen in [29]. When using spring mechanisms, the shooting power is quite hard to control. The pneumatic systems usually need a large gas tank to generate high pressure to realize strong shooting, and the number of shots generally depends on the size of the gas tank. As a result, most RoboCup MSL teams choose to use solenoid electromagnet, whose shooting force can be high and is easier to control. Our custom-designed solenoid electromagnet has been depicted in Fig. 6, it consists of a solenoid, an electromagnet core, a shooting rod, and two linear actuators with potentiometer. The shooting rod can be adjusted in height to allow for different shooting modes,

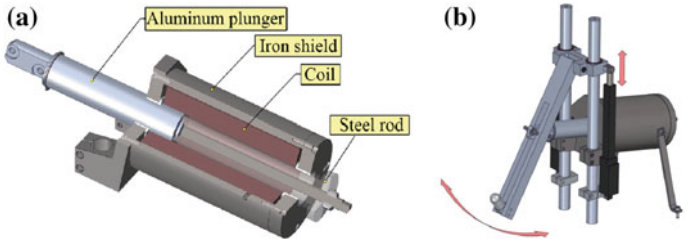

Fig. 6 **a** The solenoids electromagnet. **b** Mechanism of the shooting system

namely, flat shots for passing and lob shots for scoring. Two modes are realized using two linear actuators to move the hinge of the shooting rod to different positions. For more detail, please refer to [13].

3.2 Visual Perception System

For RoboCup MSL robots, the visual perception system is of great importance as they should be fully autonomous. There are three kind of visual perception sensors in our system, namely an omni-directional vision system, a front vision system and a RGB-D vision system. Among them, each robot has an omni-directional vision, and each regular robot has an additional front vision, while the goalie robot has dual RGB-D cameras, as shown in Fig. 3. Below a short introduction to the vision sensors is given.

Omni-Directional Vision System Almost all RoboCup MSL teams are using omni-directional vision systems, which is composed of a convex mirror and a camera pointing upward towards the mirror. The panoramic mirror plays the most important impact on the imaging quality, especially on the distortion of the panoramic image. Currently we are using the mirror designed by the team named Tech United Eindhoven [30], which has a relative simple profile, at the same time is easy to calibrate.

Front Vision The front vision system is an auxiliary sensors for the regular robots, which is a low-cost USB camera and facing down upon the ground, as shown in Fig. 3. With it, the robot can recognize and localize the ball with high accuracy when the ball is close to the robot. The position of the ball is estimated based on the pinhole camera projection model. It is of great significance for accurate ball catching and dribbling.

Dual RGB-D Cameras In the current RoboCup MSL games, most of the goals are achieved by lob shooting, so accurate estimation of the shooting touchdown-point of the ball is fundamental for the goalie robot to defend these shoots. Although the object's 3D information can be acquired using the omni-directional vision system and a front vision system together, the accuracy cannot be guaranteed because the imaging resolution of the omni-directional vision is relative low due to its large field of view (FoV). The Kinect 2 RGB-D camera can stream out color and depth information simultaneously at the frame rate of 30 fps, whose sensing range is up to 8 m. Therefore, it is an ideal sensor to obtain the 3D ball information for the goalie robot. Thus, our goalie robot is equipped with dual RGB-D cameras, as demonstrated in Fig. 3, to recognize and localize the ball, estimate its moving trace and predict the touchdown-point in 3D space.

3.3 Industrial Electrical System

As the RoboCup MSL game becomes more and more competitive and fierce, the requirements on the robustness and reliability of the electronic system are also increasing. To improve the robustness of our robot control system, the electrical system of NuBot robots is designed based on the so called PC-based control technology, whose block diagram has been drawn in Fig. 7. As can be seen, the system uses an Ethernet-based field-bus system named EtherCAT [31, 32] to realize high-speed communication between the industrial PC and the connected modules. All vision

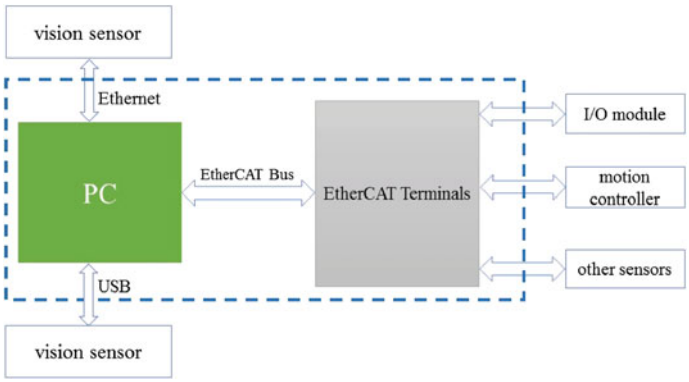

Fig. 7 The electrical system based on PC control technology. The *blue-dashed box* represents industrial PC and Ethernet-based field-bus, which are the core module of the PC-based control technology

and control algorithms are processed on the industrial PC. The industrial electrical system has been used through 2014 Brazil and 2015 China international RoboCup competitions, 2014 China RoboCup competition.

4 ROS-Based Software for NuBot Robots

The recent achievements in robotics make autonomous mobile robot play an increasingly important role in daily life. However, it is difficult to develop a generic software for different robots, e.g. it is usually difficult to reuse others' code implements. A solution named Robot Operating System (ROS), launched by Willow Garage company, provides a set of software libraries and tools for building robot applications across multiple computing platforms. ROS has many advantages: ease of use, high-efficiency, cross-platform, supporting multi-programming languages, distributed computing, code reusability, and is completely open source (BSD) and free for others to use. We also use ROS to build our NuBot software. Furthermore, our software is developed on Ubuntu, and it is also open source. For the current version, the operating system is Ubuntu 14.04, and the ROS version is indigo.

The software framework, as shown in Fig. 8, is divided into 5 main parts:

1. the Prosilica Camera node and the OmniVision node;
2. the UVC Camera node and the FrontVision node;
3. the NuBot Control node;
4. the NuBot HWControl node;
5. the RTDB and the WorldModel node.

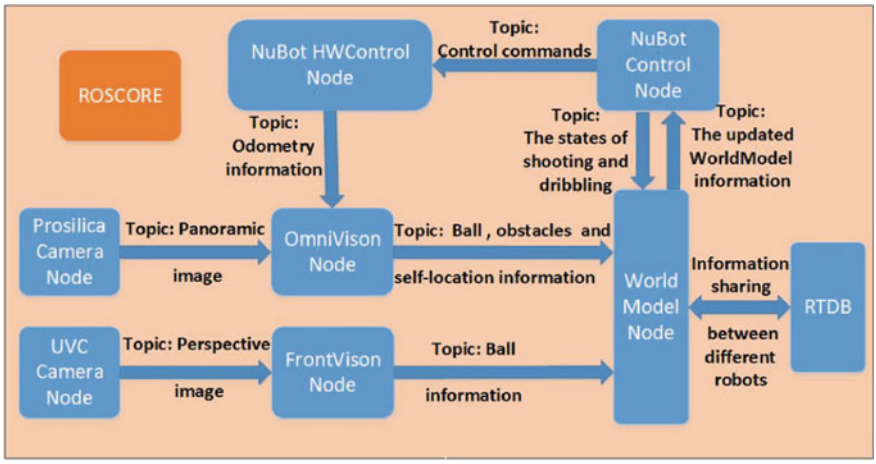

Fig. 8 The software framework based on ROS

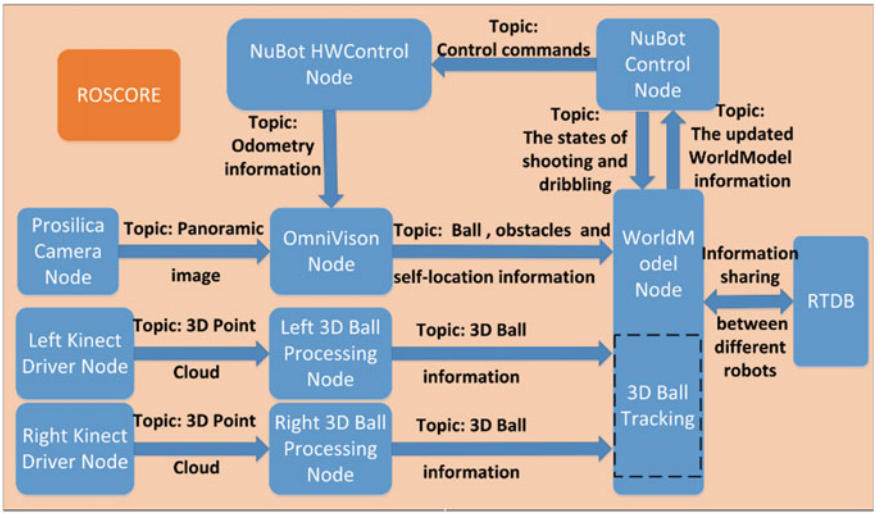

Fig. 9 The goalie software framework based on ROS

For the goalie robot, four Kinect related nodes will replace the FrontVision node and the UVC Camera node, i.e., a driver node and a 3D ball tracking node is required for each Kinect, as shown in Fig. 9. These nodes will be described in the following sub-sections.

4.1 The OmniVision Node

Perception is the base to realise the autonomous ability for mobile robots, such as path planning, motion control, self localization, action decision and cooperation. Omni-directional vision is one of the most important sensors for RoboCup MSL soccer robots. The image is captured and published by the Prosilica Camera node.[4] It takes less than 30 ms to perform the computation of below algorithms, so the OmniVision node can be run in real-time.

Colour Segmentation and White Line-Points Detection The color lookup table is calibrated off-line. Because of its simplicity and low computational requirements, it is used to realize color segmentation. A typical panoramic image captured by our omni-directional vision system is shown in Fig. 10a in a RoboCup MSL standard field, the corresponding segmentation result is shown in Fig. 10b. As can be seen, this method can be used to distinguish the ball, green field, black obstacles and white

[4] http://wiki.ros.org/prosilica_camera.

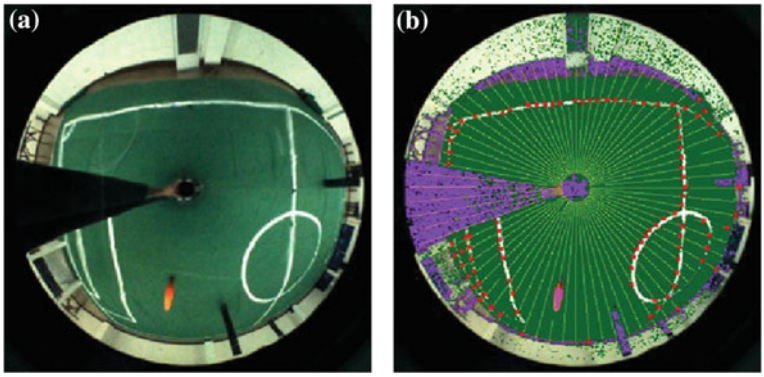

Fig. 10 **a** The image captured by our omni-directional vision system. **b** The result of color segmentation for image in **a**, for the visualization purpose, the ball has been colored with *pink* and obstacles have been colored with *purple*

line in the color-coded environment. To detect white line in the panoramic image, we search for significant color variations along some scan lines because of the different color values between the white lines and the green field. As shown in Fig. 10b, these scan lines are radially arranged around the image center, and the red points represent the resulted white line-points.

Self-localization To localize an autonomous mobile robot under a highly dynamic structured environment is still a challenge. A matching optimization algorithm has been employed to realize global localization and pose tracking for our soccer robots accurately in real-time. A brief introduction is given below, see [33] for more detail. As an off-line preprocessing step, 315 samples are acquired as the robot's candidate positions which are located uniformly in the field. Then, for the real-time global localization, the orientation is obtained by an Motion Trackers instrument (MTi). Afterwards, the match optimization localization algorithm is used to determine the real pose among the samples. Once global localized, pose tracking phase is started, where the encoders based odometry is used to obtain its coarse pose, and a Kalman Filter is employed to fuse the odometry with the match optimization result. A typical localization result is illustrated in Fig. 11, during the experiment, the robot was manually pushed to follow straight lines on the field, which is shown as black lines in the figure. The red traces depict the localization result. The mean position error is less than 6 cm.

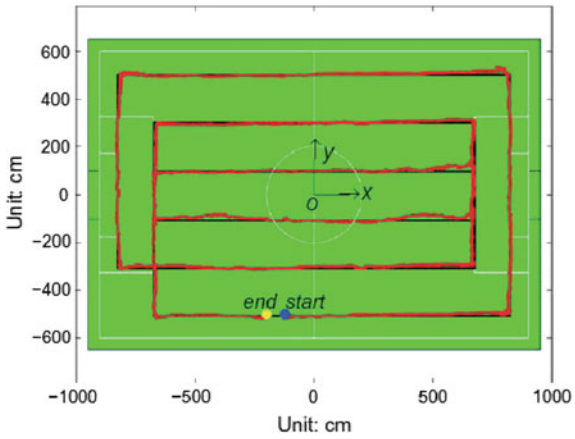

Fig. 11 The robot's self-localization results

4.2 The FrontVision Node and the Kinect Node

The FrontVision node processes the perspective image captured and published by the UVC Camera node,[5] and provides a more accurate ball position information when the ball is in the near front of a regular robot. The node detects the ball using a color segmentation algorithm and region growing algorithm similar to the OmniVision node. Then we can estimate the ball position based on the following assumptions. First, the ball is located on the ground. Second, the pinhole camera model is adopted to calibrate camera interior and exterior parameters off-line. Lastly, the height of the camera to the ground and its view direction is known.

3D information of the ball is of great significance for the goalie robot to intercept lob shot balls. However, using the front vision system and the omni-directional vision system to interpret depth information is difficult. Therefore, a dual RGB-D cameras setup is employed to recognize and localize the ball, estimating its moving trace in 3D space. The OpenNI RGB-D camera driver, which has been integrated into ROS, is employed for obtaining point clouds data in the Kinect driver node. Basic point cloud processing, such as noise filtering and segmentation, is based on algorithms of the Point Cloud Library (PCL) [34].

As shown in Fig. 12, in the 3D ball processing node, the same color segmentation algorithm as that in the OmniVision node is used to obtain some candidate ball regions. Then, the random sample consensus algorithm (RANSAC) [35] is used to fit a spherical model to the shape of the 3D candidate ball regions. With the proposed method, only a little number of candidate ball regions need to be fitted. Lastly, to intercept the ball for the goalie, the 3D trajectory of the ball regarded as a parabola curve is estimated and the touchdown-point in 3D space is also predicted in the 3D ball processing node, using a similar algorithm as in [36]. In total, the node takes

[5] http://wiki.ros.org/uvc_camera.

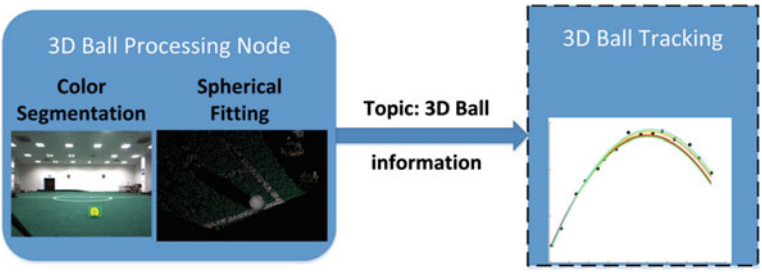

Fig. 12 The 3D ball processing data flow

about 30–40 ms to process a frame of RGB-D data, therefore can meet the real-time requirement of highly dynamic RoboCup MSL games.

4.3 The NuBot Control Node

On top level of the controllers, the NuBot soccer robots typically adopt a three-layer hierarchical structure. To be specific, the NuBot control node basically contains strategy, path planning and trajectory tracking.

The design of soccer robots aims to fulfil all the tasks completely autonomously and cooperatively. Therefore, multi-robot cooperation plays a central role in RoboCup MSL. To allocate the roles of the robots and initiate the cooperation, a group intelligence scheme is proposed to imitate the captain or the decision-maker in the competition, see [37] for detail. In our scheme, a hybrid distributed role allocation method is employed, including role evaluation, role assignment and dynamic reassignment. The soccer robot can select a proper role among the following set: attacker, defender and others. While the roles are determined, each robot is motivated to perform the corresponding tasks individually and autonomously, such as moving, defending, passing, catching and dribbling.

Path planning and obstacle avoidance is still quite a challenge under highly dynamic competition environments. To deal with it, an online path planning method based on the subtargets method and B-spline curve is proposed [38]. Benefiting from the proposed method, we can obtain a smooth path and realize real-time obstacle avoidance at a relative high speed. The method can be summarized as follows:

- generating some via-points using the subtargets algorithm iteratively;
- obtaining a smooth path by using B-spline curve between via-points; and
- optimizing the planning path via actual constraints such as the maximal size of an obstacle and the robot velocity and so on.

To track the planned path at a high speed with a quick dynamic response and low tracking error, Model Predictive Control (MPC) is utilized, as MPC can easily take into account the constraints and use the future information to optimize the current

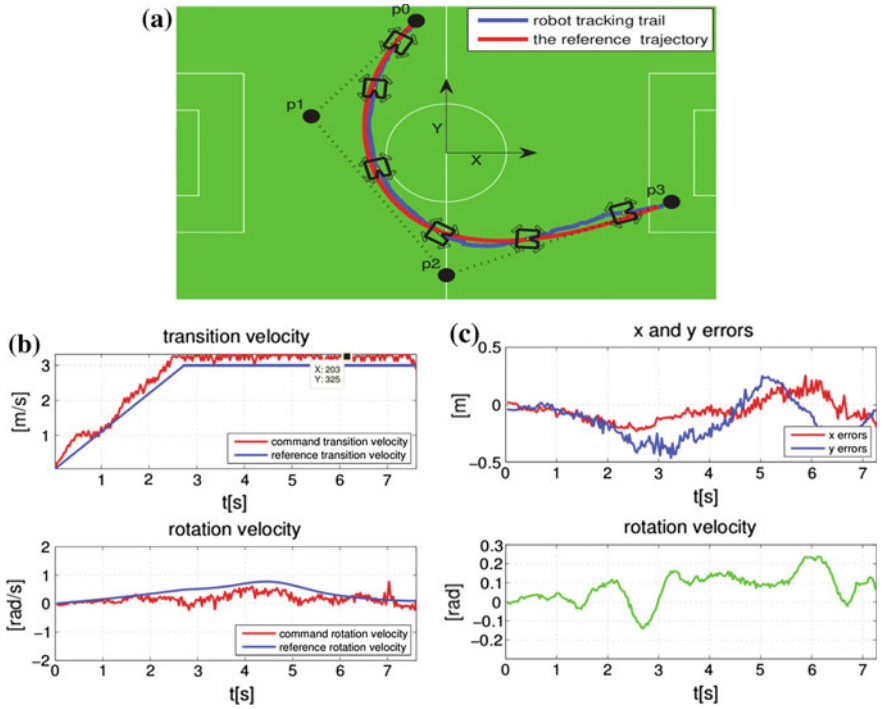

Fig. 13 A typical path tracking result of the proposed controller. **a** The robot starts at control point $p0$ to track the given trajectory, and finally stops at point $p3$. The reference trajectory and the real trajectory is shown in a *red curve* and a *blue curve*, respectively. **b** The speed during the tracking, which is bounded at 3.25 m/s. **c** The tracking errors

output [26]. Firstly, a linear full-dynamic error model based on the kinematics model of the soccer robot is obtained. Then, MPC is used to design the control law to satisfy both the kinematics constraints and kinetics constraints. Meantime, Laguerre Networks is used to design the MPC controller, in order to reduce the computational time for the online application. As illustrated in Fig. 13, the robot can track the path with a quick dynamic response and low tracking errors by our proposed MPC control law.

4.4 The NuBot HWControl Node

On bottom level of the controllers, the NuBot HWControl node performs four main tasks:

1. controlling the four motors of the base frame;
2. obtaining odometry information;

3. controlling the ball-handling system; and
4. actuating the shooting system.

The ROS EtherCAT library for our robots is developed to exchange information between the industrial PC and the actuators and sensors, e.g., AD module, I/O module, Elmo controller, motor encoder, linear displacement sensor. The speed control commands calculated in the NuBot Control node are sent to four Elmo motor controllers of the base frame at 33 Hz for realizing robot motion control. Meanwhile, the motor encoder data are used to calculate odometry information, which are published to the OmniVision node. For the third task, high control accuracy and high-stability performance are achieved by feedback plus feedforward PD control for the active ball-handing system. The relative distance between the robot and the ball measured with two linear displacement sensors is regarded as feedback signal, and the robot velocity is used as the feedforward signal. For the last task, the shooting system first needs to be calibrated off-line. During competitions, the node adjusts the hinge of the shooting rod to different heights according to the received commands: flat-shooting or lob-shooting from the NuBot Control node. Furthermore, it can determine the shooting strength according to the calibration results and kicks the ball out.

4.5 The WorldModel Node

The real-time database tool (RTDB) [22, 23] developed by the CAMBADA team is used to realize the robot-to-robot communication. The information of the ball, the obstacles and the robot itself provided by the OmniVision node, the Kinect node and the FrontVision node are combined with the data communicated from teammates to acquire a unified world representation in the WorldModel node. The information from its own sensors and other robots is of great significance for single-robot motion and multi-robot cooperation. For example, every robot fuses all obtained ball information, and only the robot with the shortest distance to ball should catch it and others should move to appropriate positions; each robot achieves accurate positions of the obstacles and obtains the positions of its teammates by communication, thus it can realize accurate teammate and opponent identification, which is important for our robots to perform man-to-man defense.

5 Gazebo Based Simulation System

In this paper, the open source simulator Gazebo [8] is employed to simulate the motions of our soccer robots. The main reason to use Gazebo is that it offers a convenient interface with ROS, which has been used to construct software for our real robots, see Sect. 4 for detail. In addition, Gazebo also features 3D simulation, multiple physics engines, high fidelity models, huge user base and etc. Therefore,

Table 1 Properties of the robot model

Property	Value
Mass	31 kg
Moment of inertia	$I_{zz} = 2.86\,\text{kg} \cdot \text{m}^2$ $I_{xx} = I_{yy} = I_{xy} = I_{xz} = I_{yz} = 0$
Friction coefficient	0.1
Velocity decay	Linear: 0. Angular: 0
Model plugin	nubot_gazebo

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 section is organized as follows. Section 5.1 introduces the creation of simulation models and a simulation world. Section 5.2 presents the realization of a single robot's basic motions by a Gazebo model plugin. Furthermore, in Sect. 5.3, the model plugin is integrated with the real robot code so that several robot models are able to reproduce real robots' behavior. Finally, in Sect. 5.4, three tests are conducted to validate the effectiveness of the simulation system.

5.1 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)[6] files. Besides, a simulation world, which determines lighting, simulation step size, simulation frequency and other simulation properties, is specified by a world file.

Simulation Models Models used in this simulation system include the NuBot robot model, the soccer field model and the soccer ball model.

- **Robot model**: It is composed of a chassis link without any joint. Table 1 lists some important properties specified in the robot model SDF file. Besides, another two important properties, mesh and collision that are used for visualization and collision detection respectively, are illustrated in Fig. 14. They are drawn by the open source 3D drawing tool SketchUp.[7] 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. Furthermore, we do not model the real robot's physical mechanisms, such as omni-directional wheels, ball-dribbling, ball-kicking and omni-vision camera mechanisms. Therefore, this model does

[6] http://sdformat.org/.

[7] http://www.sketchup.com/.

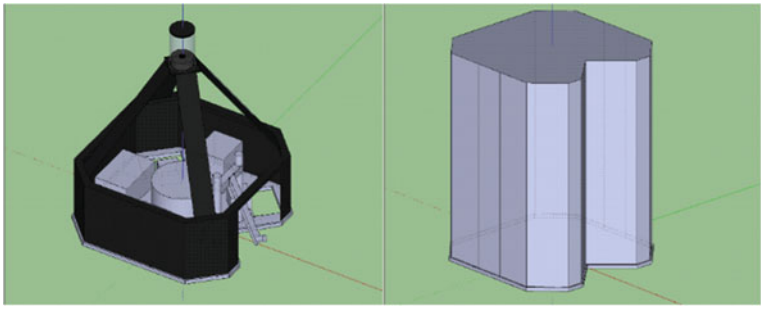

Fig. 14 Mesh and collision properties of the robot model. *Left* Mesh property; *Right* collision property

Table 2 Properties of the simulation world

Property	Value
Physics engine	Open dynamics engine
Max step size	0.007 s
Gravity	$-9.8\,\mathrm{m/s^2}$

not require any joints. The simplification is reasonable according to the simulation purpose: to test multi-robot collaboration strategies and algorithms. Therefore the emphasis of the simulation system is on the final effect of robot basic motions but not the complicated physical processes involved. The physical mechanisms capabilities are realized by a Gazebo model plugin that will be discussed in Sect. 5.2.

- **Soccer field model**: Images of the goal net, field ground and field lines, together with OGRE material scripts[8] are used to construct the field model. The field is then scaled according to the 2015 RoboCup MSL rules. The collision elements are composed of each parts' corresponding geometry.
- **Soccer ball model**: The soccer ball model is built with the same attributes of a defined FIFA (Fdration Internationale de Football Association) 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.

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 2. Finally, a simulation world with three robots and a soccer ball is created, see Fig. 15.

[8] http://www.ogre3d.org/.

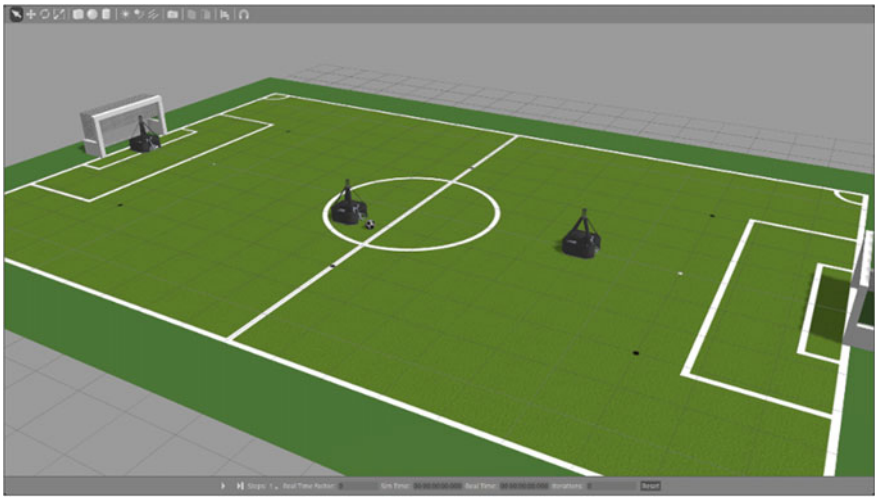

Fig. 15 The simulation world, with three robots playing a ball

5.2 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.

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 the 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.

The model plugin starts running automatically when a robot model is spawned. For example, when the robot model "bot1" is spawned, a computation graph shown in Fig. 16 is created, which is visualized by the ROS tool rqt_graph. As can be seen, 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 the /bot1/nubotcontrol/velcmd topic are used to control the robot model's velocity.

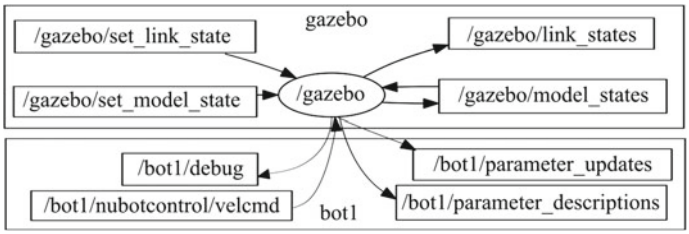

Fig. 16 The computation graph of the model plugin

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. They will be discussed in the following part.

Motion Realization A single robot's basic motions include omni-directional locomotion, ball-dribbling and ball-kicking.

- **Omni-directional 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.
- **Ball-dribbling**: If the distance between the robot and the soccer ball is within a distance threshold and the angle from the robot front direction to the ball viewing direction 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.
- **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, e.g., the ground pass and the lob shot. For the ground pass, the soccer ball does not lose contact with the ground so its initial velocity vector is calculated in the field plane. As for the lob shot, the soccer ball is kicked into the air so its speed in the up-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.

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, and RESET. The robot model performs these motions following the behavior states transfer graph as shown in Fig. 17. The test results, as shown in Fig. 18 prove that the "nubot_gazebo" model plugin realizes basic motions successfully.

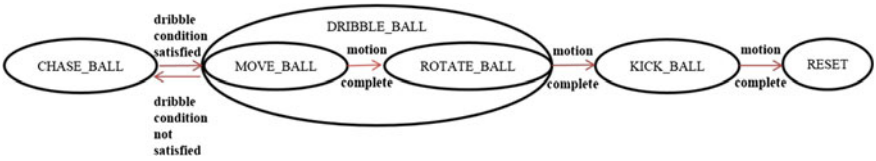

Fig. 17 Single robot behavior states transfer graph

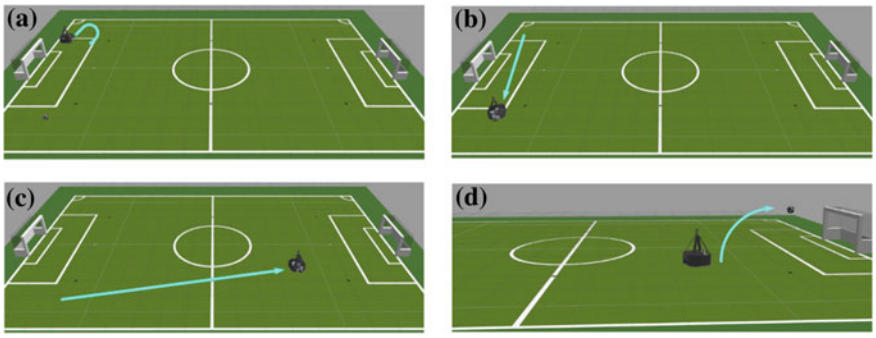

Fig. 18 Single robot simulation result. **a** Initial state; **b** CHASE_BALL state; **c** DRIBBLE_BALL state; **d** KICK_BALL state

5.3 Model Plugin and Real Robot Code Integration

It would be better to use the same interface to control the real robots and the simulated robots. In this case, the multi-robot collaboration algorithms could be evaluated using the simulation system. Furthermore, the implementation can be directly applied to the real robots without any modification. In other words, it is significant to integrate the model plugin with the real robot code.

In the real robot code, there are eight nodes in total (see Fig. 8). Among them, "world_model" and "nubot_control" are close related to multi-robot collaboration and cooperation. In addition, there is a coach program which receives and visualizes information from each robot and sends basic commands such as game-start, game-stop, kick-off and corner-ball via RTDB.

To integrate the real robot code with the model plugin, the left five nodes which are related to hardware should be replaced by 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. 19. There are three noticeable changes described as follows.

- Coach communicates with each robot's "world_model" node via ROS messages: for convenience and reliability, the communication between Coach and "world_model" no longer requires RTDB in the simulation scenario. Instead, they are able to send and receive ROS messages in one local computer. In particular,

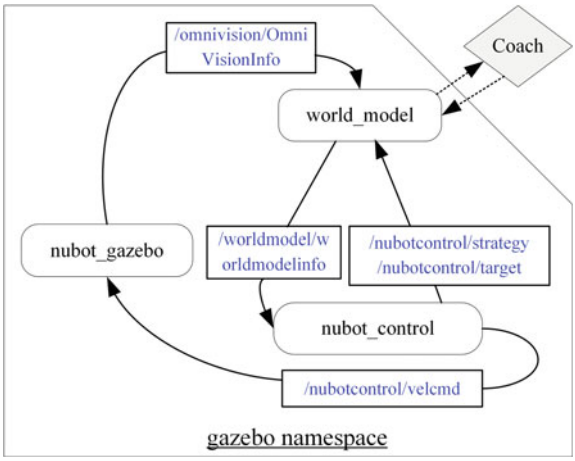

Fig. 19 The data flow graph of the integration of the real robot code and the model plugin

each robot receives messages about game status from the Coach. However, the Coach only receives the world model information from one selected robot. This is because all the robot's world model information is accurate and shared in the simulation environment, there is no need for the Coach to obtain other robots' world model information.

- An intermediary node (simulation interface) for communication among robots: in the real world scenario, robots share their own strategies information with their teammates by RTDB. However, as for simulation, it is neither practical nor necessary to use RTDB as a communication measure since all robots are simulated in one computer. Therefore, an intermediary node (simulation interface) subscribes to messages on collaboration strategies from all robots and in return, publishes new messages containing all the strategies information. So all the robots are able to share the information without the use of RTDB network communication.
  In addition, topic-name-prefixing is employed for simulation to distinguish different robots. 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 model's name to be used as a prefix to their own topic names or service names. Therefore, the robots can subscribe to their own topics or respond to their own services. These prefixes, i.e., the model names, are obtained by a bash script to guarantee that each name is mapped to the appropriate robot models as shown in Fig. 20. The bash scripts also start the simulation interface node and spawn models for Gazebo. It works as a mapping mechanism and a bridge between different separate components. This helps isolate the real robot code from the simulation components so as to improve the adaptability of the simulation system. In other words, different robot code can be easily tested in this environment since it does not depend on the simulation.
- Gaussian noise: Gaussian noise is added to the position and velocity information obtained by the robot model to mimic the real world situation.

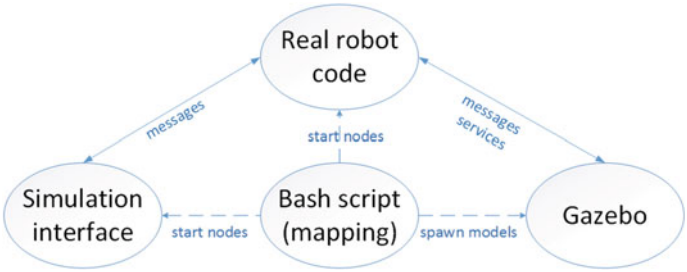

Fig. 20 The functions of the bash script

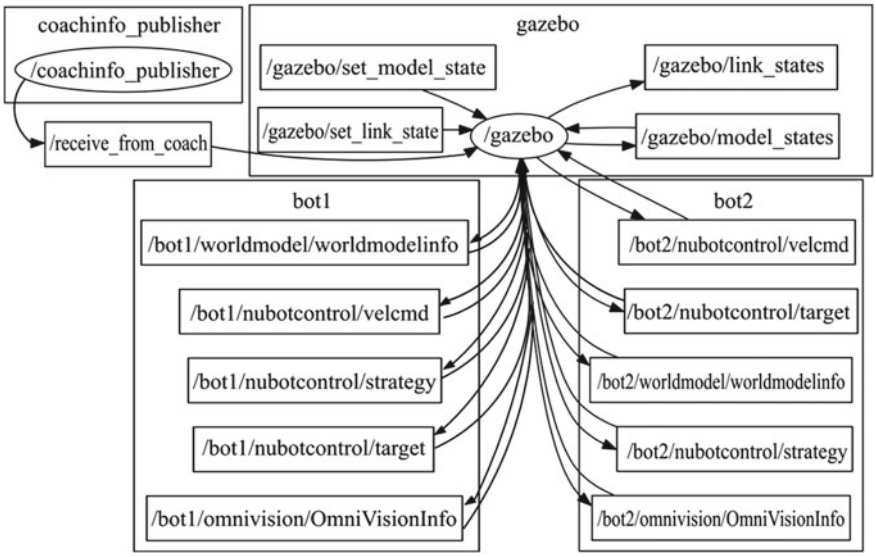

Fig. 21 The computation graph of the simulation with two robot models

Finally, two robot models bot1 and bot2 are spawned into a simulation world and the corresponding computation graph is shown in Fig. 21. Note that all the model plugins are embedded in the /gazebo node and the topic names are all prefixed by corresponding model names due to the mapping function of the bash script discussed before.

5.4 Simulation of a Match

It is also possible to simulate a match of two simulated teams, which could be used to evaluate new collaboration algorithms. Furthermore, machine learning algorithms could be used to train the simulated robots during the simulated match, and the trained

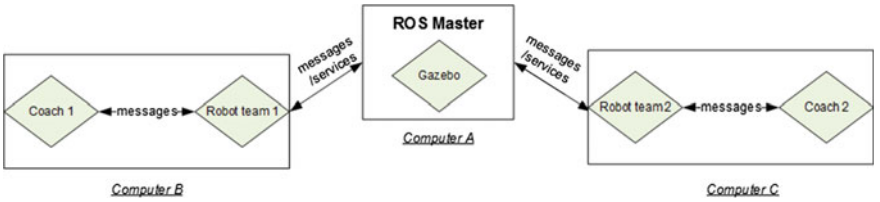

Fig. 22 The overall structure of the configuration of two simulation teams

Fig. 23 The simulation of a soccer match by two robot teams

results can be then applied to the corresponding real robots. Figure 22 shows the overall structure of the setup. There are totally three computers to simulate a soccer match between two robot teams. One of the computer is for Gazebo visualization with model plugins to simulate the motions of each robot. The other computers are used for running the real robot codes and their corresponding Coach programs. The total computation involved has been distributed to three computers and therefore, the simulation speed is fast enough to test the multi-robot coordination strategies in real time. In addition, there is only one ROS master in computer A, which registers nodes, services, topics and other ROS resources from all the three computers. Finally, the simulation of a match (without goalie) is shown in Fig. 23.

6 Single Robot Simulation Tutorial

Note that the `single_nubot_gazebo` package can simulate only ONE robot soccer player for RoboCup MSL. It is designed for demonstration of how the simulation system works. However, it can be adapted for other purposes. If you want to test multi-robot cooperation strategies, please refer to the `gazebo_visual` package, while the compilation in this tutorial is still useful. For further information, please refer to our previous paper [14].

6.1 Get the Package

If you have git installed, you could use the below command to download the package:

```
$ git clone git@github.com:nubot-nudt/gazebo_visual.git
```

As an alternation, you could also go to https://github.com/nubot-nudt/single_nubot_gazebo and download the package in zip format and extract it in your computer.

6.2 Environment Configuration

The recommended Operating Environment is Ubuntu 14.04 and ROS Jade with Gazebo included. For more operating environment, please refer to the readme file at https://github.com/nubot-nudt/single_nubot_gazebo.

ROS Jade has `gazebo_ros_pkgs` with it, so you don't have to install the package again. However, the following steps should be done to fix a bug in ROS Jade related to Gazebo:

```
$ sudo gedit /opt/ros/jade/lib/gazebo_ros/gazebo
```

In this file, go to line 24 and delete the last '/', i.e.,

```
setup_path=$(pkg-config --variable=prefix gazebo)/share/gazebo/
```

is replaced with

```
setup_path=$(pkg-config --variable=prefix gazebo)/share/gazebo
```

After these steps, try to run the command below to check if it is successful.

```
$ rosrun gazebo_ros gazebo
```

or

```
$ roslaunch gazebo_ros empty_world.launch
```

If either one is successful running, then you are ready for the following steps.

6.3 Package Compiling

(1) Go to the package root directory (`single_nubot_gazebo`), e.g.

```
$ cd ~/single_nubot_gazebo
```

(2) If you already have `CMakeLists.txt` in the `src` folder, then you can skip this step. If not, execute the commands below:

```
$ cd src
$ catkin_init_workspace
$ cd ..
```

(3) Configure the package using the command below. In this step, you may encounter some errors related to Git. However, if you did not use Git, just ignore them.

```
$ ./configure
```

(4) Compiling the package, the simulation system is ready if the compiling is completed done.

```
$ catkin_make
```

6.4 Package Overview

The robot movement is realized by a Gazebo model plugin called `NubotGazebo` generated by source files `nubot_gazebo.cc` and `nubot_gazebo.hh`. Most importantly, the essential part of the plugin is realizing three motions: omni-directional locomotion, ball-dribbling and ball-kicking.

Basically, this plugin subscribes to topic `/nubotcontrol/velcmd` for omni-directional movement, and subscribes to services `/BallHandle` and `/Shoot` for ball-dribbling and ball-kicking, respectively. You can customize this code for your robot based on these messages and services as a convenient interface. The types and definitions of the topics and services are listed in Table 3.

For the definition of `/BallHandle` service, when `enable` equals to a non-zero number, a dribble request would be sent. If the robot meets the conditions to dribble the ball, the service response `BallIsHolding` is true. For the definition of `/Shoot service`, when `ShootPos` equals to -1, this is a ground pass. In this case, `strength` is the initial speed you would like the soccer ball to have. When `ShootPos` equals to 1, this is a lob shot. In this case, `strength` is useless since the strength is calculated by the Gazebo plugin automatically and the soccer ball would follow a parabola path to enter the goal area(only if the robot heads towards

Table 3 Topics and services

Topic/Service	Type	Definition

/nubotcontrol/velcmd	nubot_common/VelCmd	`float32 Vx`
		`float32 Vy`
		`float32 w`

/BallHandle	nubot_common/BallHandle	`int64 enable`
		`---`
		`int64 BallIsHolding`

/Shoot	nubot_common/Shoot	`int64 strength`
		`int64 ShootPos`
		`---`
		`int64 ShootIsDone`

the goal area). If the robot successfully kicks the ball even if it failed to goal, the service response `ShootIsDone` is true.

There are three ways for a robot to dribble a ball, e.g.,

(a) Setting ball pose continually: this is the most accurate one; nubot would hardly lose control of the ball, but the visual effect is not very good (the ball does not rotate).
(b) Setting ball secant velocity: this is less accurate than method (a) but more accurate than method (c).
(c) Setting ball tangential velocity: this is the least accurate. If the robot moves fast, such as 3 m/s, it would probably lose control of the ball. However, this method achieves the best visual effect under low-speed condition.

For package `single_nubot_gazebo`, it uses method (c) for better visualization effect. However, for package `nubot_gazbeo`, it uses method (a) for better control of the soccer ball.

6.5 Single Robot Automatic Movement

The robot will do motions according to the state transfer graph shown in Fig. 17, following the below steps:

(1) Go to the package root directory.
(2) source the setup.bash file:

```
$ source devel/setup.bash
```

(3) Using roslaunch to load the simulation world

```
$ roslaunch nubot_gazebo sdf_nubot.launch
```

Note: Step 2 should be performed every time to open a new terminal. Alternatively, this command can be wrote into the `~/.bashrc` file so that step 2 is not required when opening new terminal.

Finally, the robot will rotate and translate with a given trajectory, i.e., it accelerates at a constant acceleration and stays at a constant speed after reaching the maximum velocity.

You could click the `Edit->Reset World` from the menu (or press ctrl-shift-r) to reset the simulation world so the robot would do the basic motions again.

When the robot reaches its final state (HOME), its motion can be controlled using keyboard under `$ rosrun nubot_gazebo nubot_teleop_keyboard`. You could also run `$ rqt_graph` to see the data flow chart of messags/topics.

6.6 NuBotGazebo API

For the detailed information and usage of the `NubotGazebo` class, please refer to the `doc/` folder.

6.7 How You Could Use It to Do More Stuff

The main purpose of the simulation system is to test multi-robot collaboration algorithms. As a precondition, the users have to know how to control the movement of each robot in the simulation. The topic publishing and service calling could be inferred by reading the source of keyboard controlling. In a word, to control the movement of the robots requires publishing velocity commands on the topic `/nubotcontrol/velcmd`. If the robot is close enough to the ball, dribble the ball by calling the ROS service named `/BallHandle` and kick the ball by calling the service named `/Shoot`. The types and definitions of theses topics and services are presented in Table 3.

7 Multi Robot Simulation Tutorial

7.1 Package Overview

The following three packages should be used together to simulate multi-robots together. The `nubot_ws` and the `conach4sim` package can be downloaded at https://github.com/nubot-nudt/nubot_ws and https://github.com/nubot-nudt/coach4sim respectively.

package	description
`gazebo_visual`	For robot simulation and visualization
`nubot_ws`	For robot controlling
`coach4sim`	Game command sending

Qt has to be installed in order to use `coach4sim`. However, for those who do not want to install Qt, a solution is to use ROS command line tools for sending game commands:

```
  $ rostopic pub -r 1 /nubot/receive_from_coach
nubot_common/CoachInfo"
    MatchMode: 10
    MatchType: 0"
```

In the command, MatchMode is the current game command, MatchType is the previous game command. The coding of the game commands is in core.hpp. For quick reference:

```
enum MatchMode {
          STOPROBOT  =  0,
          OUR_KICKOFF = 1,
          OPP_KICKOFF = 2,
          OUR_THROWIN = 3,
          OPP_THROWIN = 4,
          OUR_PENALTY = 5,
          OPP_PENALTY = 6,
          OUR_GOALKICK = 7,
          OPP_GOALKICK = 8,
          OUR_CORNERKICK = 9,
          OPP_CORNERKICK = 10,
          OUR_FREEKICK = 11,
          OPP_FREEKICK = 12,
          DROPBALL     = 13,
          STARTROBOT   = 15,
          PARKINGROBOT = 25,
          TEST = 27
        };
```

The robot movement is realized by a Gazebo model plugin which is called NubotGazebo generated by source files nubot_gazebo.cc and nubot_gazebo.hh. Basically the essential part of the plugin is realizing basic motions: omni-directional locomotion, ball-dribbling and ball-kicking.

The plugin single_nubot_gazebo is similar to that in package single_nubot_gazebo, i.e., it subscribes to the topic nubotcontrol/velcmd for omnidirectional movement, and subscribes to the service BallHandle and Shoot for ball-dribbling and ball-kicking, respectively. For package gazebo_visual, there is a new topic named omnivision/OmniVisionInfo which contains messages about the soccer ball and all the robots' information such as position, velocity and etc. Since there may be multiple robots, the name of those topics and services should be prefixed with the robot model names in order to be distinguished with each other. For example, if a robot model's name is nubot1, then the topic names are /nubot1/nubotcontrol/velcmd and /nubot1/omnivision/OmniVisionInfo and the service names would be /nubot1/BallHandle and /nubot1/Shoot accordingly. The types and definitions of the topic nubot1/omnivision/OmniVisionInfo is as:

```
Header header
BallInfo ballinfo
ObstaclesInfo obstacleinfo
RobotInfo[] robotinfo
```

As shown above, there are three new message types in the definition of the omnivision/OmniVisionInfo topic, i.e., BallInfo, ObstaclesInfo and RoboInfo. The field robotinfo is a vector. Before introducing the format

of these messages, three other underlying message types `Point2d`, `Ppoint` and `Angle` are listed below.

```
# Point2d.msg, representing a 2-D point.
float32 x               # x component
float32 y               # y component

# PPoint.msg, representing a 2-D point in polar coordinates.
float32 angle          # angle against polar axis
float32 radius         # distance from the origin

# Angle.msg, representing the angle
float32 theta           # angle of rotation

# BallInfo.msg, representing the information about the ball
Header header          # ROS header defined in std_msgs
int32 ballinfostate    # the state of the ball information
Point2d pos            # position in the global reference
                       # frame
PPoint real_pos        # relative position in the robot
                       # body frame
Point2d velocity       # velocity in the global reference
                       # frame
bool pos_known         # ball position is known(1) or not(0)
bool velocity_known    # ball velocity is known(1) or not(0)

# ObstaclesInfo.msg, representing the obstacles information
Header header          # ROS header defined in std_msgs
Point2d[] pos          # position in the global reference
                       # frame
PPoint[] polar_pos     # position in the polar frame, whose
                       # origin is the center of the robot
                       # and the polar axis
                       # is along the kicking mechanism

# RobotInfo.msg, representing teammates' information
Header header          # ROS header defined in std_msgs
int32 AgentID          # ID of the robot
int32 targetNum1       # robot ID to be assigned for target
                       # position 1
int32 targetNum2       # robot ID to be assigned for target
                       # position 2
int32 targetNum3       # robot ID to be assigned for target
                       # position 3
int32 targetNum4       # robot ID to be assigned for target
                       # position 4
int32 staticpassNum    # in static pass, the passer's ID
int32 staticcatchNum   # in static pass, the catcher's ID
Point2d pos            # robot position in global coordinate
                       # system
Angle heading          # robot heading in global coordinate
                       # system
float32 vrot           # rotational velocity in the global
                       # coordinate system
Point2d  vtrans        # linear velocity in the global
```

```
                        # coordinate system
bool iskick             # robot kicks the ball(1) or not(0)
bool isvalid            # robot is valid(1) or not(0)
bool isstuck            # robot is stuck(1) or not(0)
bool isdribble          # robot dribbles the ball(1) or not(0)
char current_role       # the current role
float32 role_time       # time duration that the robot keeps
                        # the role unchanged
Point2d  target         # target position
```

7.2 Configuration of Computer A and Computer B

The recommended way to run simulation is with two computers to run `nubot_ws` and `gazebo_visual` separately, i.e., computer A runs `gazebo_visual` to display the movement of robots, while computer B runs `nubot_ws` to control the virtual robots. In addition, computer B should also run the coach program for sending game commands. Communication between computer A and computer B is via ROS topics and services.

Following is an configuration example:

- Adding each other's IP address in the `/etc/hosts` file;
- Run `gazebo_visual` in computer A;
- In computer B, export `ROS_MASTER_URI` and then run `nubot_ws`;
- In computer B, run the `coach` and send game command.

8 Conclusion

In summary, we have presented the ROS based software and Gazebo based simulation for our RoboCup MSL robots. ROS based software makes it easier to share data and code among RoboCup MSL teams, and construct hybrid teams. Further, we have also detailed the design of the interface between the robot software and simulation, which brings the possibility to evaluate multi-robot collaboration algorithms using the simulation.

We expect this work to be of value in the RoboCup MSL community. On the one hand, the researchers can refer to our method to design both software and simulation for RoboCup MSL robots, or even general robots. On the other hand, the NuBot simulation software can be used to simulate RoboCup MSL matches, which enables the state-of-the-art machine learning algorithms to be used for multi-robot collaboration training.

Lastly, the presented ROS based software and Gazebo based simulation can also be employed for multi-robot collaboration researches more than RoboCup with little modification.

Acknowledgements Our work is supported by National Science Foundation of China (NO. 61403409 and NO. 61503401), China Postdoctoral Science Foundation (NO. 2014M562648), and graduate school of National University of Defense Technology. All members of the NuBot research group are gratefully acknowledged.

References

1. Kitano, H., M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. 1997. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, 340–347. ACM.
2. Kitano, H., M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. 1997. Robocup: A challenge problem for AI. *AI Magazine* 18 (1): 73.
3. Kitano, H., M. Asada, I. Noda, and H. Matsubara. 1998. Robocup: robot world cup. *IEEE Robotics Automation Magazine* 5: 30–36.
4. Almeida, L., J. Ji, G. Steinbauer, and S. Luke. 2016. *RoboCup 2015: Robot World Cup XIX*, vol. 9513. Heidelberg: Springer.
5. Bianchi, R.A., H.L. Akin, S. Ramamoorthy, and K. Sugiura. 2015. *RoboCup 2014: Robot World Cup XVIII*, vol. 8992. Heidelberg: Springer.
6. Soetens, R., R. van de Molengraft, and B. Cunha. 2014. Robocup msl-history, accomplishments, current status and challenges ahead. In *RoboCup 2014: Robot World Cup XVIII*, ed. R.A.C. Bianchi, H.L. Akin, S. Ramamoorthy, and K. Sugiura, 624–635. Heidelberg: Springer.
7. Rohmer, E., S.P.N. Singh, and M. Freese. 2013. V-rep: A versatile and scalable robot simulation framework. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1321–1326.
8. Koenig, N., and A. Howard. 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings*, vol. 3, 2149–2154. IEEE.
9. Michel, O. 1998. Webots: Symbiosis between virtual and real mobile robots. In *the First International Conference on Virtual Worlds*, (London, UK), 254–263. Springer.
10. Der, R., and G. Martius. 2012. The LpzRobots Simulator. In *The Playful Machine Ralf*, ed. R. Der, and G. Martius. Heidelberg: Springer.
11. Harris, A., and J.M. Conrad. 2011. Survey of popular robotics simulators, frameworks, and toolkits. In *2011 Proceedings of IEEE Southeastcon*, 243–249.
12. Castillo-Pizarro, P., T.V. Arredondo, and M. Torres-Torriti. 2010. Introductory survey to open-source mobile robot simulation software. In *Robotics Symposium and Intelligent Robotic Meeting (LARS), 2010 Latin American*, 150–155.
13. Xiong, D., J. Xiao, H. Lu, Z. Zeng, Q. Yu, K. Huang, X. Yi, Z. Zheng, C. Loughlin, and C. Loughlin. 2016. The design of an intelligent soccer-playing robot. *Industrial Robot: An International Journal* 43 (1).
14. Yao, W., W. Dai, J. Xiao, H. Lu, and Z. Zheng. 2015. A simulation system based on ros and gazebo for robocup middle size league. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 54–59. IEEE.
15. Van De Molengraft, M., and O. Zweigle. 2011. Advances in intelligent robot design for the robocup middle size league. *Mechatronics* 21 (2): 365.
16. Nadarajah, S., and K. Sundaraj. 2013. A survey on team strategies in robot soccer: team strategies and role description. *Artificial Intelligence Review* 40 (3): 271–304.
17. Nadarajah, S., and K. Sundaraj. 2013. Vision in robot soccer: a review. *Artificial Intelligence Review* 1–23.
18. Li, X., H. Lu, D. Xiong, H. Zhang, and Z. Zheng. 2013. A survey on visual perception for RoboCup MSL soccer robots. *International Journal of Advanced Robotic Systems* 10 (110).

19. Nardi, D., I. Noda, F. Ribeiro, P. Stone, O. von Stryk, and M. Veloso. 2014. Robocup soccer leagues. *AI Magazine* 35 (3): 77–85.
20. Lunenburg, J., R. Soetens, F. Schoenmakers, P. Metsemakers, R. van de Molengraft, and M. Steinbuch. 2013. Sharing open hardware through rop, the robotic open platform. In *Proceedings of 17th annual RoboCup International Symposium*.
21. Neves, A.J., A.J. Pinho, A. Pereira, B. Cunha, D.A. Martins, F. Santos, G. Corrente, J. Rodrigues, J. Silva, J.L. Azevedo, et al. 2010. *CAMBADA soccer team: from robot architecture to multiagent coordination*. INTECH Open Access Publisher.
22. Santos, F., L. Almeida, P. Pedreiras, and L.S. Lopes. 2009. A real-time distributed software infrastructure for cooperating mobile autonomous robots. In *International Conference on Advanced Robotics, 2009. ICAR 2009*, 1–6. IEEE.
23. Santos, F., L. Almeida, and L.S. Lopes. 2008. Self-configuration of an adaptive TDMA wireless communication protocol for teams of mobile robots. In *IEEE International Conference on Emerging Technologies and Factory Automation, 2008. ETFA 2008*, 1197–1204. IEEE.
24. Lu, H., S. Yang, H. Zhang, and Z. Zheng. 2011. A robust omnidirectional vision sensor for soccer robots. *Mechatronics* 21 (2): 373–389.
25. Lu, H., H. Zhang, J. Xiao, F. Liu, and Z. Zheng. 2008. Arbitrary ball recognition based on omni-directional vision for soccer robots. In *RoboCup 2008: Robot Soccer World Cup XII*, ed. L. Iocchi, H. Matsubara, A. Weitzenfeld, and C. Zhou, 133–144. Heidelberg: Springer.
26. Zeng, Z., H. Lu, and Z. Zheng. 2013. High-speed trajectory tracking based on model predictive control for omni-directional mobile robots. In *2013 25th Chinese Control and Decision Conference (CCDC)*, 3179–3184. IEEE.
27. Xiao, J., H. Lu, Z. Zeng, D. Xiong, Q. Yu, K. Huang, S. Cheng, X. Yang, W. Dai, J. Ren, et al. 2015. Nubot team description paper 2015. In *Proceedings of RoboCup 2015, Hefei, China*.
28. Rajaie, H., O. Zweigle, K. Häussermann, U.-P. Käppeler, A. Tamke, and P. Levi. 2011. Hardware design and distributed embedded control architecture of a mobile soccer robot. *Mechatronics* 21 (2): 455–468.
29. Zandsteeg, C. 2005. Design of a robocup shooting mechanism. University of Technology Eindhoven.
30. Martinez, C.L., F. Schoenmakers, G. Naus, K. Meessen, Y. Douven, H. van de Loo, D. Bruijnen, W. Aangenent, J. Groenen, B. van Ninhuijs, et al. 2014. Tech united eindhoven, winner robocup 2014 msl. In *Robot Soccer World Cup*, 60–69. Springer.
31. Jansen, D., and H. Buttner. 2004. Real-time ethernet: the ethercat solution. *Computing and Control Engineering* 15 (1): 16–21.
32. Prytz, G. 2008. A performance analysis of EtherCAT and PROFINET IRT. In *IEEE International Conference on Emerging Technologies and Factory Automation, 2008. ETFA 2008*, 408–415. IEEE.
33. Xiong, D., H. Lu, and Z. Zheng. 2012. A self-localization method based on omnidirectional vision and mti for soccer robots. In *2012 10th World Congress on Intelligent Control and Automation (WCICA)*, 3731–3736, IEEE.
34. Rusu, R.B., and S. Cousins. 2011. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), 9–13 May 2011.
35. Schnabel, R., R. Wahl, and R. Klein. 2007. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum* 26 (2): 214–226.
36. Lu, H., Q. Yu, D. Xiong, J. Xiao, and Z. Zheng. 2014. Object motion estimation based on hybrid vision for soccer robots in 3d space. In *Proceedings of RoboCup Symposium 2014*, (Joao Pessoa, Brazil).
37. Wang, X., H. Zhang, H. Lu, and Z. Zheng. 2010. A new triple-based multi-robot system architecture and application in soccer robots. In *Intelligent Robotics and Applications*, ed. H. Liu, H. Ding, Z. Xiong, and X. Zhu, 105–115. Heidelberg: Springer.
38. Cheng, S., J. Xiao, and H. Lu. 2014. Real-time obstacle avoidance using subtargets and Cubic B-spline for mobile robots. In *Proceedings of the IEEE International Conference on Information and Automation (ICIA 2014)*, 634–639. IEEE.

Author Biographies

Junhao Xiao (M'12) received his Bachelor of Engineering (2007) from National University of Defense Technology (NUDT), Changsha, China, and his Ph.D. (2013) at the Institute of Technical Aspects of Multimodal Systems (TAMS), Department Informatics, University of Hamburg, Hamburg, Germany. Then he joined the Department of Automatic Control, NUDT (2013) where he is an assistant professor on Robotics and Cybernetics. The focus of his research lies on mobile robotics, especially on RoboCup Soccer robots, RoboCup Rescue robots, localization, and mapping.

Dan Xiong is a Ph.D. student at National University of Defense Technology (NUDT). He received his Bachelor of Engineering and Master of Engineering both from NUDT in 2010 and 2013, respectively. His research focuses on image processing and RoboCup soccer robots.

Weijia Yao is a Master student at National University of Defense Technology (NUDT). He received his Bachelor of Engineering from NUDT in 2015. He curretly focuses on multi-robot coordination and collaboration.

Qinghua Yu is a Ph.D. student at National University of Defense Technology (NUDT). He received his Bachelor of Engineering and Master of Engineering both from NUDT in 2011 and 2014, respectively. His research focuses on robot vision and RoboCup soccer robots.

Huimin Lu received his Bachelor of Engineering (2003), Master of Engineering (2006) and Ph.D. (2010) from National University of Defense Technology (NUDT), Changsha, China. Then he joined the Department of Automatic Control, NUDT (2010) where he is an associate professor on Robotics and Cybernetics. The focus of his research lies on mobile robotics, especially on RoboCup Soccer robots, RoboCup Rescue robots, omni-directional vision, and visual SLAM.

Zhiqiang Zheng received his Ph.D. (1994) from University of Liege, Liege, Belgium. Then he joined the Department of Automatic Control, National University of Defense Technology where he is a full professor on Robotics and Cybernetics. The focus of his research lies on mobile robotics, especially on multi-robot coordination and collaboration.