Design and Implementation of a Simulated Robot Using Webots Software

Velappa Ganapathy¹ Lui Wen Lik Dennis²

¹School of Engineering, Monash University Malaysia
1 Jalan Kolej, Bandar Sunway, Petaling Jaya, Selangor Malaysia
Tel: +60-3-5636-0600, Fax: +60-3-5636-0622, Email: velappa.ganapathy@eng.monash.edu.my

²School of Engineering, Monash University Malaysia
2 Jalan Kolej, Bandar Sunway, Petaling Jaya, Selangor Malaysia
Tel: +60-3-5636-0600, Fax: +60-3-5636-0622, Email: dwllui@gmail.com

Abstract – Robot navigation remains as one of the unsolved problems of computer science and robotics. The problem is in the development of an algorithm which would allow the mobile robot to find its way around a room, or maze and at the same time dealing with any obstacles. Researchers had further decomposed the problem into several modules in view of the navigation problem. These are such as obstacle avoidance, wall following, path planning, etc. Acquiring robots in the facility incurs cost. Generally, it will not be cheap. Thus, it may be costly to perform tests on the real robot if it is damaged during its operation. In addition, the robot needs to be handled carefully and calibrations need to be performed. As such, simulators have come to play an important role to verify these algorithms before it is tested on the real robot. In this paper, a flexible simulation is developed for the Khepera II robot using Webots. The developed simulation is equipped with various controls to allow the user to setup different experiments. Upon completion, developed controllers will be tested in the simulation. This includes the Braitenberg and Neural Network Reinforcement Learning Algorithms. Then, the results are verified by implementing the controller on the real robot.

Keywords – simulation, Webots commercial mobile simulation software, Khepera II.

I. INTRODUCTION

According to the Encarta Dictionary, simulation is defined as the reproduction of essential features of something, the imitation or feigning of something or the construction of a mathematical model to reproduce the characteristics of a phenomenon, system, or process, often using a computer, in order to infer information to solve problems. Historically, simulations used in different fields had been developed independently. However, modern simulation development that uses computers had led to some unification and a more systematic view of the concept. Simulations have been widely utilized for training purposes such as flight simulators and educational purposes. In addition, it plays an important role in engineering systems or any system that involves many processes. Most engineering simulations entail mathematical modeling and computer assisted investigation. However, depending on application, there are cases where mathematical modeling is not reliable.

The exponential growth in the field of robotics has led to the existence of a new breed of robots in the market. Apart from the familiar industrial manipulators which are heavily used in automated facilities such as assembly lines, welding of car components or sorting soda cans, a new category of robots was brought to the market. The new category of robots is known as the mobile robots. Mobile robots have evolved so significantly that it has reached the level of everyday usage. According to the United Nations Economic Commission for Europe [1], at the end of year 2004, there were currently one million industrial robots in service and the number of household mobile robots total up to 600,000. Some of these household robots are such as the lawnmower robot and vacuum cleaner robot shown in Figure 1.

![Fig. 1. (a) Lawnmower Robot, (b) Vacuum Cleaner Robot](image-url)

It is an inevitable fact that the cost associated with the development of robots is huge. As such, simulations play a rather crucial role in this field such that development costs are minimized. By simulating the robot and its environment in a virtual world, the designer is able to approach the design in a much more optimum manner taking into consideration of the design and control parameters of the robot in the simulated environment. In addition, researchers could analyze the behavior of the designed controllers in the simulation to avoid damage on the real one. Thus, this motivates the development of various mobile robot simulator programs [2-3]. However, to
allow experiments to be conducted in the simulation, it will require the modeled environment to sufficiently represent the actual one. Otherwise, simulation results will not be valid. On top of that, to facilitate a wide variety of experiments to be performed using the simulation in an efficient manner, a certain degree of flexibility should be incorporated into the simulation such that the user could manipulate the environment and robot easily for experimental purposes.

In this paper, an enhanced robot simulator program with a certain degree of flexibility is discussed. It is developed for the Khepera II using Webots [3], which is a commercial mobile robot simulation software developed by Cyberbotics Ltd. The rest of this paper is organized as follows: Section II discusses the core features of Webots and reveals some related projects using Webots. Section III discusses the developed simulation environment and its features. Section IV presents the results. Finally, conclusions are drawn in Section V.

II. WEBOTS

Webots is software used by over 250 universities and research centers worldwide to model, program and simulate mobile robotics. The main reason for its increasing popularity is due to its ability to reduce the overall development time. Development time is reduced via a fast prototyping and simulation for any mobile robot, including wheel, legged and flying robots. It includes a complete library of sensors and actuators. Hence, developers could easily include these features on the robot by simply treating these sensors and actuators as objects in the environment. In addition, it allows the user to program the robot through C, C++ and Java or from third party software such as Matlab and LabVIEW through TCP/IP.

Simulating complex robotics devices such as articulated mechanical parts requires precise physics simulation. This is provided by Webots via the Open Dynamics Engine (ODE). In addition, the supervisor capability offered by Webots allows experiments to be supervised by dynamically moving objects, sending messages to robot, record robot trajectories, add new objects or robots, etc. Once programming is completed, the controllers can be transferred to the real robots such as Aibo, Lego Mindstorms, Khepera and Hemisson to evaluate and compare the actual behavior of the robot with its simulated behavior.

Many projects were successfully developed utilizing Webots. However, only the more significant ones will be discussed. To investigate on metric navigation, especially on the creation and usage of grid occupancy, Szabó, R. [4] has utilized this platform in his project and successfully modeled and simulated the Khepera I in Webots. The author has successfully demonstrated a working method on how to use occupancy grids to efficiently navigate in Webots. Similarly, Magyar, B. et al [5] have utilized the platform to evaluate its control algorithm, known as the “Zax” algorithm, which has been consistently participating in the ALife (Artificial Life) contest. This is the place where intelligent mobile robot controller algorithms compete against one another. The proposed algorithm consists of six main modules: Search, Charger Approach, Battery Charge, Distant Outer Wall Approach and Wall Follow. As such, the Webots environment is not only ideal for researching and developing controller algorithms but it also provides an opportunity for researchers and enthusiasts to compare their algorithms in this contest.

On the other hand, Hohl, L. [6] and Cominoli, P. [7], both under the supervision of Prof. Auke Jan Ijspeert and Dr. Olivier Michel has successfully modeled and simulated the Aibo four legged robot dog from Sony and Fujitsu HOAP-2 humanoid robot using Webots. Various control schemes were designed for the robots. These control schemes are then evaluated on the simulation environment. Finally, it is validated by cross-compiling the controllers into the real robot. Figure 2 shows the Fujitsu HOAP-2 robot.

A similar project which involves the use of Webots to develop Khepera I applications could be seen in the work by Wang, L.F. et al [8]. In comparison, the simulation discussed in this paper is developed for the Khepera II and it is much more flexible. Nevertheless, this paper is a solid reference to the work being discussed in this paper. In addition, Webots was also used as an experimental platform by Handa, H. et al [9] to test their proposed algorithm which consists of an ordinal Fuzzy Classifier System and the Genetic Algorithm.

III. Modeling the Robot and its Environment

The simulation environment in Webots is entirely based on the Virtual Reality Modeling Language (VRML). As such, there are numerous features in Webots which resembles the features of the VRML nodes. More accurately, it is the same features but these features are represented in a tree structure. This tree structure is known as the scene tree. The scene tree contains all information necessary to describe the graphical representation and simulation of the 3D world. It is composed of a list of nodes, each containing fields which may contain values or nodes. Nodes could be included, discarded and manipulated very easily using the controls. As the scene tree is altered, the codes in the ‘.wbt’ file which is generated by Webots changes accordingly. The codes in the ‘.wbt’ file are actually VRML codes with minor differences. In other words, Webots allow the user to write VRML codes via the construction of the scene tree. The simulated robot and environment has to be built from scratch. Upon opening a new file in Webots, a basic template which consists of a pointlight...
node and checkerboard floor is loaded. Figure 3 shows the Webots development environment and Figure 4 shows an overview of the developed system.

Once the robot is modeled, it will be exported into VRML97 format and then imported into Webots under the DifferentialWheels node. To complete the Khepera II in Webots, 8 infrared and light sensors will be placed around the robot.

A. Modeling the Khepera II Mobile Robot

In this project, the Khepera II was to be simulated. It was selected over the ActiveMedia Pioneer robot which is available in our laboratories mainly because the Khepera II is supported by Webots. This makes cross-compilation or remote control schemes to be implemented straightforwardly. Moreover, the Khepera II is a very popular tool as it was designed as a research and teaching tool in the framework of a Swiss Research Priority Program. Although Webots supports the design of complex objects through a flexible set of nodes using points, lines and faces, it would still be a very time consuming task. As such, 3DS Max was utilized in the design of the Khepera II as shown in Figure 4. Other 3D modelers could also be used as long as it is capable of exporting models into the VRML97 format. Only simple objects which could be built by using primitive shapes are designed directly in Webots. Figure 5 shows the Khepera II model being designed in 3ds Max.

B. Modeling the Simulation Environment

Upon successful completion of the Khepera II model, walls are built such that the robot does not escape into 3D space. All four sides of the walls are built using boxes. Although the walls could be graphically seen on the computer screen, however, it does not prevent the robot from escaping. To prevent it from escaping, bounding objects has to be defined for each object in the environment which includes the Khepera II itself. Bounding objects are only limited to primitive shapes such as box, cylinder and sphere.

Up to this stage, a simple environment is built and the robot has been included into the simulation environment. The product of this is as shown in Figure 6. All objects are static in the environment except for the robot. However, without the controller program, the robot will not move. A static environment is not desirable. Thus, to incorporate flexibility into the simulation, the supervisor and robot controller programs were developed. This will be discussed in the following subsections.

C. The Supervisor Controller Program

The supervisor has executive powers being restricted only by the simulation environment. It is possible of performing more than what the robot can do. The supervisor has the ability to move and rotate any objects in the simulation, sending and receiving messages via the ‘Receiver’ and ‘Emitter’ nodes and track the coordinates of objects. In short, it overlooks the entire simulation with certain limitations. In addition, it is capable of taking snapshots during simulation. These pictures could be saved as ‘.jpg’ or ‘.png’ formats. As such, flexibility could
only be achieved if the supervisor controller program is written for the simulation.

The supervisor program can only be programmed in C or C++. These languages do not support the development of Graphical User Interface (GUI). As such, Microsoft Visual C.Net 2002 (MSVC) [10] was utilized due to the GUI development support offered through its Managed Extensions C++ (MC++). MC++ is compatible with both C and C++ languages and supports multithreading. The GUI developed for the supervisor program is as shown in Figure 7.

![Fig. 7. Supervisor GUI (a) Tabpage1, (b) Tabpage2, (c) Tabpage3](image)

There are various features incorporated into this GUI. These features are provided to allow the user to customize experiments in a fast and efficient manner. Some features could be implemented straightforwardly, while others may require some programming tweaks. The features are:

(i) Repositioning & reorientation of sensors

The controls located in Figure 7(a) and (b) allow the user to reorientate and reposition the sensors around the circumference of the Khepera II easily. Thus, the user will not be required to trace their way through the entire scene tree and perform manual calculations if relocation of the sensors is required. These personal settings could be saved and retrieved for future use. Figure 8 shows the range of rotation and position for each sensor.

![Fig. 8. Range of Rotation and Position of Each Sensor](image)

(ii) Tracking robot position

This feature is implemented straightforwardly as it is directly supported by the supervisor. The x, y and z coordinates of the robot will be retrieved and displayed on the text fields located on the right of Figure 7(a).

(iii) Custom Maze

The custom maze feature provides the user with the flexibility for designing custom mazes by simply clicking and dragging the cursor on top of a specially designed interface as shown in Figure 9.

![Fig. 9. Custom Maze](image)

To allow the construction of a maze by the user in real time, blocks of various sizes, depending on the floor design, will be required. Thus, the inclusion of these additional blocks in the simulation environment will mean that either these blocks are already present in the simulation environment during startup, or it is to be included using the import function. Unfortunately, to position the blocks in the 3D environment will require these blocks in the environment to be initialized in the supervisor program during startup. Hence, it is not possible to implement this feature utilizing the import function as all imported blocks will have the same features and identifier (name field). The solution to this setback is to have these cubes present in the simulation environment. The user will not realize the presence of these blocks as they are located far away from the robot and its environment. As the number of blocks to be initialized is huge, the initialization process of all the blocks have to be spread over a few simulation steps to avoid buffer overflow. After the maze is designed, it could be saved for future retrieval.

(iv) Specifying Wall and Floor Textures

Wall and floor textures play an important role when it comes to vision based controllers. The flexibility to alter the wall and floor textures will facilitate the investigation of vision based controllers. Unfortunately, this useful feature is not provided handily in Webots. To change the texture of an object in Webots, the user will be required to manually find their way to the appearance node of an object in the scene tree. Then, a different URL is specified for the image to be loaded. Thus, if the environment is made up of 300 objects, the user has to do this 300 times. In view of the importance of this feature, the developed supervisor program will be capable of texture manipulation. The key concept here is to have all these wall
objects to point to a single image. Let’s call this image ‘wallImage’. Whenever the user would like to change the texture of the walls, the address of the new image is specified by the user. Then, a duplicate copy of the new image is made and renamed as ‘wallImage’. Finally, the original ‘wallImage’ is replaced by this new ‘wallImage’ and simulation is refreshed. A simulation with an altered wall and floor texture is shown in Figure 10.

(v) Light intensity specification

Vision based controllers are greatly affected by the variations in light intensity. Researchers around the world are proposing robust controllers which are based on techniques that are insensitive towards the variation in light intensity. However, this problem is still prevalent. Thus, allowing changes to be made to the light intensity values in the environment enables the user to test the sensitivity of their algorithm towards light intensity variations. Although the effect in light intensity changes is directly supported by Webots and could be observed in the environment once it is altered, however, this change in light intensity will not affect the camera image. The camera image does not update itself upon the new light intensity value as it is initialized with a fixed intensity value during startup. The solution to this problem is to utilize a ‘.dat’ file. Every time the supervisor controller starts up, it looks for this ‘.dat’ file and applies this value to the environment prior to the initialization of the camera.

(vi) Importing Additional Objects

The developed supervisor controller program also allows the user to introduce several pre-designed objects into the environment. The controls could be found in Figure 7(c). On top of that, it has the capability to introduce more than one Khepera II to the simulation environment. This feature is very useful for testing multi-robot algorithms. Figure 11 shows the introduction of some new objects in the simulation environment.

(vii) Supervisor – Robot Controller Communication

In fact, the supervisor controller and robot controller could operate at complete isolation. However, there are various features which require the interface between the supervisor and robot controller. These features will be discussed in the robot controller program. To realize the communication between the supervisor and robot controllers, the ‘Receiver’ and ‘Emitter’ nodes will be utilized. These nodes are intentionally developed to model the infra-red or radio emitter on board a robot. Thus, it acts as virtual communication channels in the simulation which could be utilized for inter-robot communication or supervisor to robot controller communication.

D. Robot Controller Program

The core component of the robot controller program is the navigation algorithms. However, this does not mean that further flexibility could not be incorporated into this program. Similarly, MC++ from MSVC was used to develop the GUI. It wraps around the navigation algorithms which are programmed in C and C++. Through this interface, the user is able to switch from one algorithm to another. The GUI developed for this program is as shown in Figure 12.

The functions available on the robot controller program are:

(i) Display panel

The display panel in Figure 12(a) allows the user to monitor the sensor and encoder readings in the simulation. The top view of the Khepera is drawn by overriding the onPaint method to retrieve a Graphics object from the argument PaintEventArgs*. An additional checkbox is available to stop this continuous updating process.

(ii) Robot trajectory

Sometimes, it might be useful to observe the path taken by the robot. This could be achieved by drawing its path on the floor as it navigates around the designated area. The pen node was used to provide this function. This node could be activated using the checkbox in Figure 12(a) and the color of the pen could be selected from the combo box. However, once the pen is activated, it could not be switched off. To switch off the pen, an invisible color is applied to the pen. As such, it appears as if it is turned off to the user. Figure 13 shows the robot trajectory.
Camera and gripper modules

The basic Khepera II only comes with 8 infrared and light sensors and a set of wheel encoders. This configuration only facilitates low-level behavior algorithms. To expand the usage of the Khepera, there are additional camera and gripper modules which allow researchers to experiment with a wide variety of algorithms which requires the input from a color/monochrome/linear grayscale camera and output to a gripper module. As these modules are very useful for research purposes, it has become very popular. Thus, it will be very attractive to include this feature into the simulation such that the gripper and camera modules could be included with no extra effort from the user.

Similar to the blocks used in the maze, the gripper and camera module is present in the simulation during startup such that it could be initialized. As the user requests for the inclusion of these modules in the simulation, the robot controller program will signal the supervisor program. Upon receiving the signal, the supervisor program will then reposition the required module on the top cover of the simulated Khepera II.

In addition, there are extra controls included for the camera and gripper module in the simulation. These controls could be found in Figure 12 (b) and (d). Using these controls, the user is able to define the Field of View (FOV) of the camera and as well as capturing and saving an image from the camera at that particular instant. The gripper controls provided allow the user to manually command the gripper to grip and release. This function is not as useful as the other functions. Nevertheless, it is able to provide the user with an overview of the grippers’ operation. Figure 14 shows the inclusion of the gripper and camera module in the simulation.

Navigation algorithms

Various navigation algorithms have been developed. However, it will not be discussed in detail in this paper. These algorithms include the Braitenberg algorithm and the Neural Network Reinforcement Learning algorithms. The user is able to swap from one algorithm to another using the controls in Figure 12(c). In addition, the user could also activate and deactivate the various sensors on the robot using this set of controls. The Braitenberg algorithm is programmed in C++ and the Neural Network Reinforcement Learning algorithms are programmed partly using C and Matlab 7.1.

C-programs to Matlab Interface

C programs have the capability to invoke Matlab sessions. By establishing the communication with Matlab through a C program, it would allow the utilization of various advance tools such as the Neural Network toolbox and Image Processing toolbox in Matlab. However, running Matlab as a background computation engine will increase the computational requirements of the processor. As a result, the user will experience a drop in simulation speed. The overall reduction in performance is totally dependant on the processor. This interface is tested by transferring color images from the camera in the simulation to Matlab. The result of the image transfer to Matlab is successful and it is illustrated in Figure 15.
However, connecting the robot through this manner will be very troublesome as the robot has to be located near to the laptop with the serial cable hanging in the air. As such, the radio turret and base were used to replace the serial cable. Webots do not support this wireless communication because this part of the software is hard-coded, and the radio base requires the command ‘*1’ to be sent through the serial communication to the radio base once it is started up such that the connection could be established with the robot with radio ID 1. The solution to this problem is to use a set of Matlab routines, kMatlab, which is developed by Piguet, Y. and Legon, S. [11]. It includes Windows DLLs to perform the system level serial port communication and a library of useful Matlab m-files to read sensors, set speed, etc. Then, through simple Matlab programming, the command ‘*1’ could be sent to the radio base.

Once the wireless communication is up and running, the simulation could be validated. Firstly, the behavior of the robot in the simulation is recorded. Then, in wireless mode, the path taken by the robot is also recorded. It is expected that if the simulated and actual robot shares the same initial position, the paths taken by the both robots should be close. From this experiment, it reveals that the simulation needs to be refined. The remote control strategy results in a noticeable motion mismatch between the simulated and actual robot. Furthermore, the percentage of deviation of the infrared sensors which is modeled based on Figure 15 is unknown. As such, a small experiment was conducted to measure the encoder readings of one step actions taken by the simulated and real robot, and the sensor readings taken with respect to the distance from a white paper are recorded. Finally, the simulation is refined and validated by comparing it with the actual robot. Figure 16 shows the configuration of the wireless mode on the real robot.

V. CONCLUSION

In this paper, a simulation for the Khepera II mobile robot with a certain degree of flexibility was developed. Some of the notable features in the simulation are the custom maze design feature, repositioning and reorientation of the sensors, changing wall and floor textures, light intensities, etc. In addition, it has interfaced Webots with the MSVC integrated development environment by using a combination of MC++, C++ and C programs. Furthermore, the C programs were interfaced to Matlab 7.1 through the Matlab engine. Matlab was further used to communicate to the serial port such that control commands could be sent to the real robot and vice versa via the radio base and turret. In conclusion, the developed simulation was not only successfully used as a platform to test the Neural Network Reinforcement Learning algorithms but it has also proved to be useful for the experiments of other algorithms with the amount of features incorporated into the simulation through the supervisor and robot controller programs.

ACKNOWLEDGMENT

The authors thank Monash University Malaysia for the support of this work.

REFERENCES