Development of a Mobile Robotic Simulator

K. Kelly, P. Wardlaw, C. McGinn

Department of Mechanical and Manufacturing Engineering, Trinity College Dublin

ABSTRACT

Robotic simulators facilitate design and testing of robotic platforms. For the purpose of rapid development and education, currently available simulation packages can be overly complex and time consuming for the purpose of rapid development and education. The aim of this work reported here was to develop a robotic simulator that would address these issues.

The programming language python was chosen for its ease of use, readability, and rapid development pedigree. Third party software libraries were used develop the simulator; wxPython - a library for building graphical user interfaces; Pygame - a library used for 2d image rendering and user interaction.

The simulator in its current state provides good functionality for the testing, development and validation of robotic control systems. Using the developed robotic simulator it is possible to simulate simple worlds, test the performance of robotic control algorithms, graphically visualise simulations in 2d, and perform probabilistic mapping.

KEYWORDS: Robotic, Simulation

1. INTRODUCTION

Robotic simulation is the process of simulating robots in a virtual environment for the purposes of test and design of automated systems. In the case of mobile robotics a large area of interest is navigation – localisation, perception, path planning, object detection, and collision prevention. Robotic simulation is a form of continuous system simulation rather than discreet [1]. Navigation cannot always be broken down into a straight forward series of unique events as surroundings tend to change.

The use of robotic simulation during a robot's development can greatly increase the quality and cost effectiveness of its design. Simulation can be used to validate control algorithms and to help engineers better visualise their algorithm's behaviours. When compared to physical testing, simulation also allows for more rapid development and reductions in time and money required [2].

2. ROBOTIC SIMULATION

Robotic simulators started to appear in the early 90s [1]. The first two companies to develop software were Deneb Robotics (now part of Dassault Systems [3]) and Tecnomatix Technologies (now a subsidiary of Siemens [4]). Originally robotic simulation was only for robotic arms [1], and was focused on manufacturing and automation system such as welding robots or pallet handling robots.

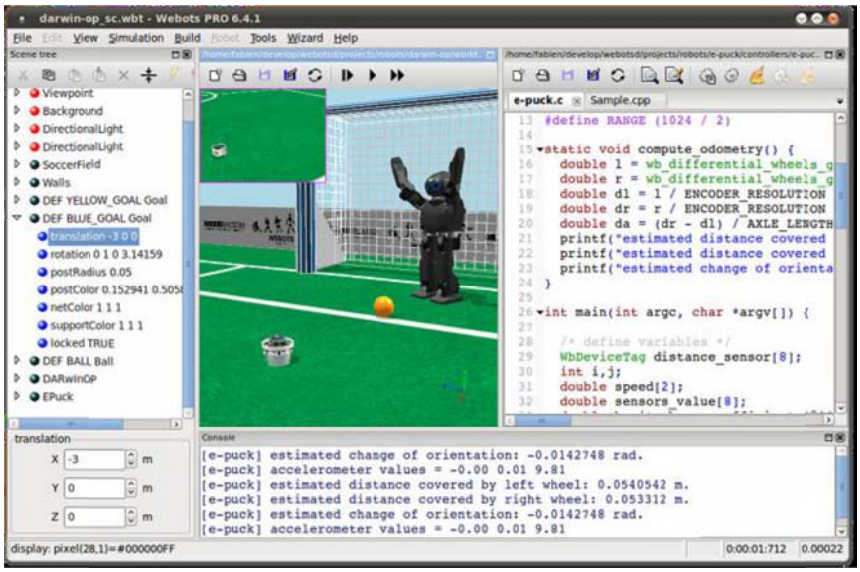

Figure 1: Webots – a popular robotic simulator. [5]

Mobile robotic simulators started appearing in the mid 90s. In 1996 Webots – a robotic software suite used to model, program and simulate virtual robots - was released [5]. Webots was created by Olivier Michel at the Swiss Federal Institute of Technology. In 1998, Michel created a spinoff company Cyberbotics which continues to develop Webots. Nowadays there are many other robotic simulation and development solutions available commercially or for free [6].

There is a good selection of commercial mobile robotic simulation packages currently available. The general trend is to provide an integrated development environment where robot designers can develop their virtual robot and environment, and test their robot control programs. Commercial simulators provide a comprehensive tool set allowing users to develop their AI code and robot chassis design. The following features are generally incorporated:

- Modelling facilities for creating virtual robots
- Programming interfaces for many languages (Java, C++, python, MATLAB, URBI)
- Included APIs providing useful programming functions.
- Powerful Graphics and Physics capabilities for providing real time simulation.
- Integrated environments facilitating whole project control.
- Large libraries of prebuilt sensors, environments, robots and other objects.

As well as commercial solutions there are many open source projects for mobile robotic simulators. Open source projects tend to primarily run on Linux, although very often are cross platform and support all major operating systems. The Player project is an open source project that *"enables research in robot and sensor systems"* [7]. Robotic simulation sees a lot of attention from enthusiast and experts in the open source community.

3. THEORY

In this section some key concepts covered in this project shall be explained.

3.1 Kinematic and Sensor Modelling

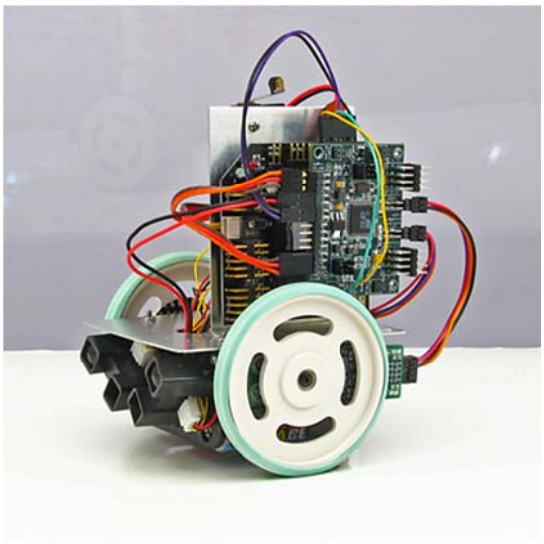

Figure 2: Differentially driven two wheeled robot, Picture source [8]

For the purposes of this project it was required to model simple robotic chassis types. In the simulator reported upon here, a 2 wheeled, differentially driven chassis model is implemented. The mathematics used was taken from source [9].

Other important kinematic models are: the Ackerman steering model (automobiles), Omni-directional wheels (Swedish/Caster wheels), serpentine and legged robots . These models were outside the scope of this project, but can be implemented into the reported simulator at a later stage. For more information on mentioned models please see reference [9].

Figure 3: SICK LIDAR sensor, [10]

The sensors modelled in the reported robotic simulator were LIDAR, Ultrasonic and Infrared range finding sensors. These sensors work on the principles of time of flight (LIDAR and Ultrasonic) and optical triangulation

(Infrared range sensors). User-customisable Gaussian error models were implemented which allow the calculation of, for each sensor type, the probability that its readings were correct, given a sensor reading value. We shall see that this parameter is very important for probabilistic mapping.

3.2 Probabilistic Mapping

Probabilistic mapping is used in mobile robotics to construct a map or image of an environment using sensor readings in order for robots to perform localisation, path planning and collision avoidance. Three key components to probabilistic mapping are occupancy grids, Bayes' theorem and Gaussian distributions.

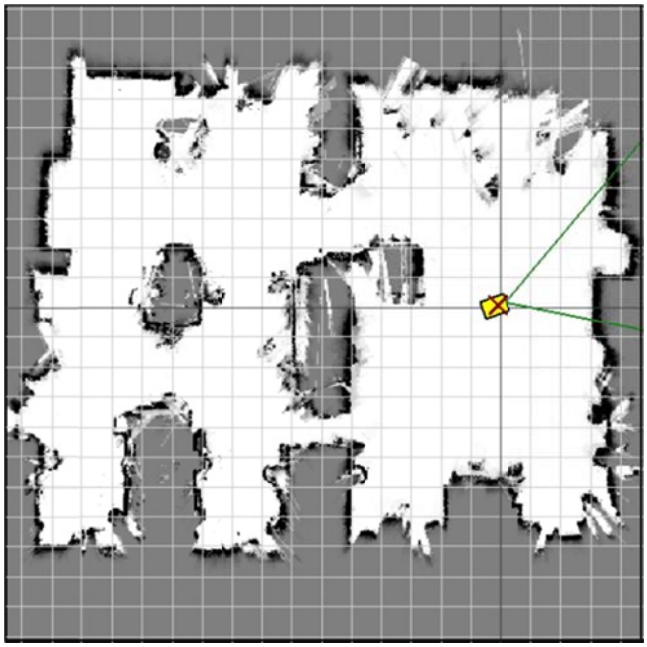

Figure 4: Occupancy grid produced by simultaneous localisation and mapping [11]

Occupancy grid maps are spatial representations of an environment. They represent an environment using a grid, each grid element containing information that reflects the occupancy of that element. Occupancy grids facilitate various key aspects of mobile robot navigation such as localisation, path planning, and collision avoidance. Occupancy grids are commonly referred to as belief spaces.

Bayes' theorem is a theorem of probability originally stated by Reverend Thomas Bayes. Bayes' theorem is fundamental to Bayesian statistics, and has applications in fields including science, engineering, medicine and law. The Bayesian interpretation expresses how a subjective degree of belief should rationally change to account for evidence.

In probabilistic mapping we use Bayes' theorem to assess the likelihood of an object existing at a specific location, given a sensor reading and some existing information. Each sensor reading taken has a corresponding degree of certainty which affects the hypothesis that an obstacle exists at a given location.

4. PROJECT OVERVIEW

Clearly there are many commercial and open source simulation suites currently available. So why not just buy current available commercial software? Existing solutions tend to be overly complicated and time consuming for the purpose of rapid development and education. In creating a robotic simulator one can:

1. Have full control of what the software can do
2. The potential to address/explore issues that exist with current simulation packages
3. As a learning exercise – To gain a higher understanding of mobile robotic navigation.

The core aims of the simulator are to:

1. Be accessible for users of any skill level. Anyone from a robotic software engineer to budding robotic enthusiast should be able to use the simulator effectively.
2. Significantly decrease in the lead time and complexity of trivial tasks such as setting up a simulation, compared to current commercial and open source solutions currently available.
3. To be an invaluable aid in the process of designing and assessing navigation, path planning, perceptive and localisation robotic control algorithms.
4. To be easily adaptable and expandable for future development.
5. To combine a high level API for controlling robots that is universally adaptable.

5. CREATING THE SIMULATOR

5.1 Programming

Obviously making a significant piece of software like this requires significant programming so much thought went into selecting the language, tools and libraries that were used to build the simulator. The programming language of choice was python. Python is a high level, interpreted, dynamically typed programming language. Python allows for rapid development of easily readable, cross platform code.

Python has many uses and applications in science and engineering. An example of the use of python in mechanical and manufacturing engineering is SciPy. SciPy combines several mathematical libraries for numerical modelling, scientific calculations, plotting, and information analysis [12].

5.2 Design

The overall structure of the simulator is composed of 5 main components: the Graphical User Interface, the Control Interface, the simulator world, and the probability spaces – shown in figure 5 below.

The world class contains all of the robots, objects and environments required for a simulation. The world class provides various functions for the addition, interaction, and sharing of objects contained within. The world is the core component of a simulator, and is frequently updated and queried by other

components of the simulator. The world class also facilitates the creation of new objects.

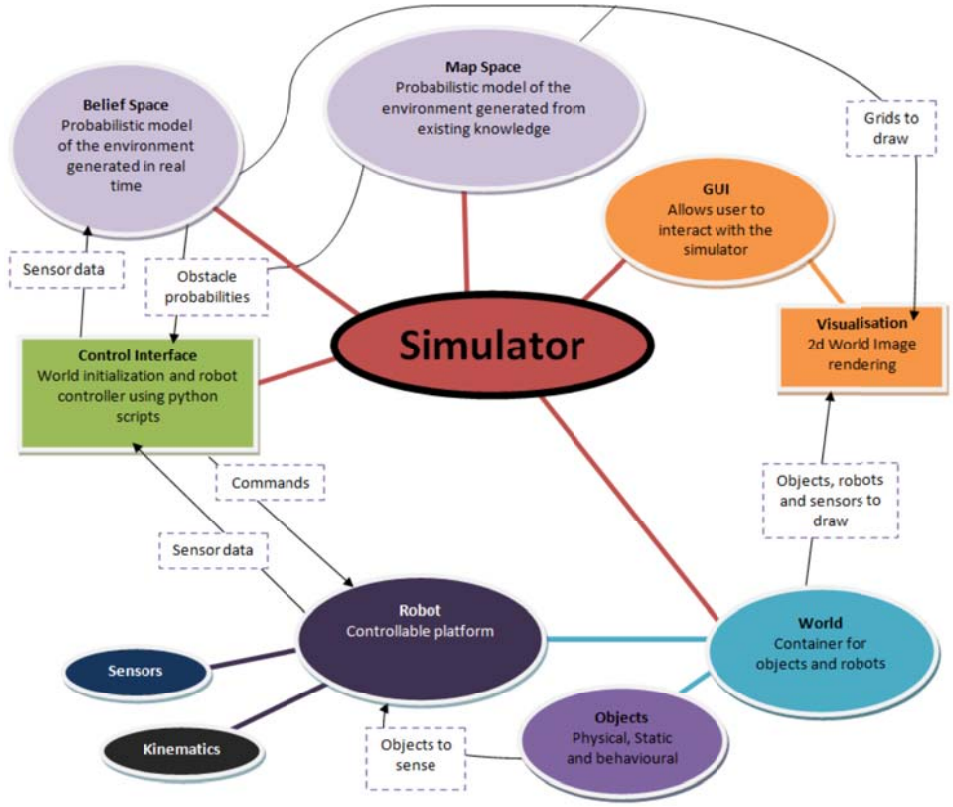

Figure 5: Simulator Structure

The Robot class is a flexible platform capable of housing any arrangement of sensors, and be driven by any form of locomotion system (differentially driven, robotic legs, Ackerman, etc). The Robot class inherits the Basic Object class so that its shape, position and orientation may be easily described and manipulated. The Robot class inherits the python thread object so that it may run its own system functions asynchronously (in a non blocking manner) from the rest of the simulator's processes. In a way the robot class can be thought of having its own "unique CPU" which it can run functions on without affecting or being affect by the rest of the simulator.

The GUI contains several major components which provide the user with visual, text, and numerical feedback, and allow the user to interact with the simulator. Unlike most of the other classes in this project, the GUI component relies heavily on external libraries (wxPython and Pygame).

The main GUI class acts to glue several components of the user interface together, and facilitate passing data between them and other components of the simulator. Each component is its own contained class aiding modularity, code reusability, and program stability (if one component fails it may not affect other components).

In the developed simulator there are two probability spaces: the belief space and the map space. The purpose of the belief space is to map the robots surroundings based upon sensor readings. This mapped data can then be used by path planning and localisation algorithms. The purpose of the map space is to map all known static features of the world. The map space describes our existing knowledge of an environment, whereas the belief space can be used to build a probabilistic image without any prior knowledge.

5.3 Functionality

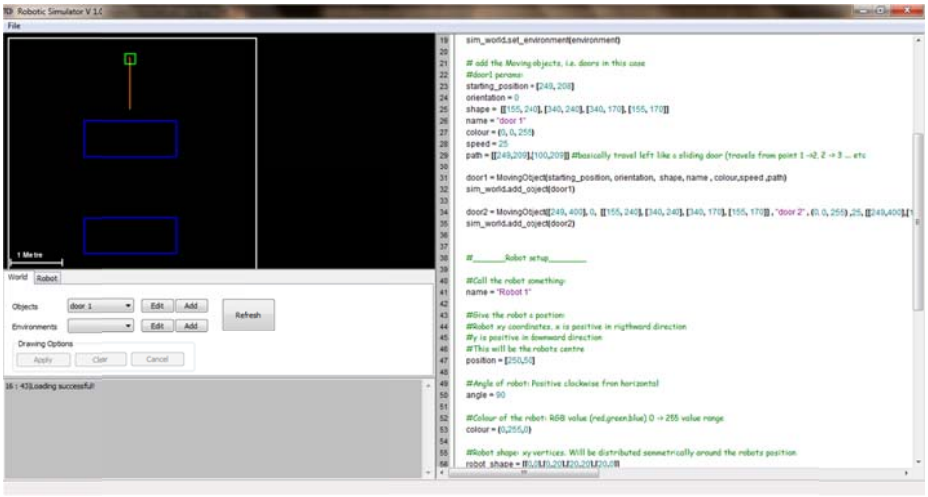

Figure 6: The developed simulators user interface.

The simulator can be used to effectively simulate a wide range of scenarios for the purpose of testing mobile robotic navigation and control algorithms.

Objects and robotic components that are modelled include: basic static and moving obstacles, flexible robot platform capable of multiple arrangements, sensors – LIDAR, Ultrasound, infrared, and chassis kinematic models – such as 2 wheel differentially driven chassis.

Using the simulator one may graphically observe their tests in real time. The simulator is able to visualise a defined simulation world, as well as probabilistic grids. Trivial tasks such as drawing an object/setting up a test scenario can be done quickly and easily using the simulators interface and dialogues.

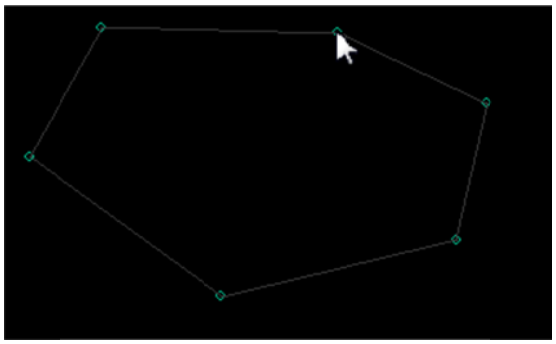

Figure 7: Editing object vertices via direct mouse input

Robot control files may be edited using the simulators code panel. Key words, comments, function names and data types are automatically displayed in a corresponding font/style. This code layout is similar to other integrated programming environments such as Microsoft visual studio, MATLAB, or Eclipse. The user written code interacts with their simulated robotic hardware via a predefined API.

```
13 def run(robot, keep_going, belief_space, map_space, interact):
14     "A simple control routine"
15
16     robot("Robot 1").start()
17     robot("Robot 1").set_wheel("left",8)
18     robot("Robot 1").set_wheel("right",8)
19
20     test_ended = False
21     belief_space.set_robot(robot("Robot 1"))
```

Figure 8: Simulator Code Interface

Finally as previously mentioned the simulator facilitates probabilistic mapping and visualisation to aid users in their development and understanding of their control algorithms. Two probabilistic spaces are included: the belief space and the map space. The inbuilt belief space can be used by a robotic controller to produce a probabilistic mapping an unknown area using data gathered from sensors. This acquired information can be used to aid localisation and path planning algorithms. The inbuilt map space allows the user to specify gird maps of an environment for use by localisation and path planning algorithms.

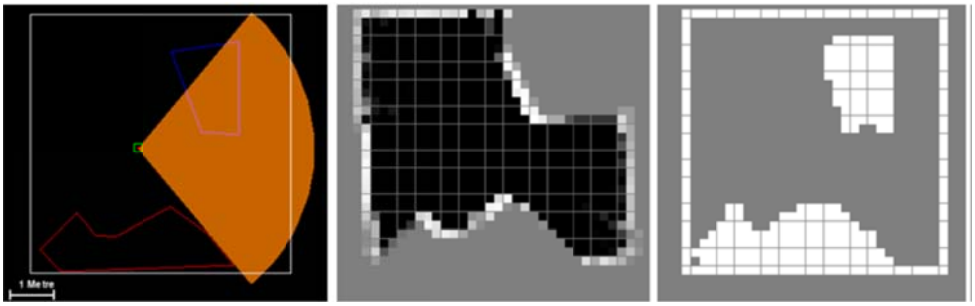

Figure 9: Left: an environment being mapped. Centre: view of the mapped environment in the belief space. Right: Discretized view of the environment in map space.

6. FUTURE WORK

There are many additional features that could be added to the simulator to provide increased functionality. Currently the simulator only accommodates 2 dimensional simulation and visualisation; 3d visualisation of the virtual world

would improve user comprehension, and could be used as a powerful demonstration tool. The addition of 3d kinematics would allow for other robotic applications such as the design of flying robots, or multi axis robot arms, to be performed using the developed simulator.

Further predefined models would allow for testing of more varied systems. Content that could be added includes: more kinematic models - Ackerman, legged, omni-directional, and more sensors models -bumpers, computer vision, gyroscopes, compasses, etc.

Figure 10: Robot swarm, picture source [13]

Swarm robotics is a branch of robotics interested in systems that consist of large numbers of mostly simple robots. Swarm robotics is akin to the field of artificial swarm intelligence, as well as the biological studies of insects, ants and other fields in nature, where swarm behaviour occurs. The developed simulator could be easily modified to support multiple virtual robotic platforms simultaneously, allowing for testing of swarm artificial intelligence.

7. CONCLUSION

Robotic simulation clearly has an important role in the development of robotic platforms and control strategies, but can be over complicated for use in rapid development and education. It can be said that there is a skill barrier preventing new users from becoming involved in the field of robotic simulation.

With the desire to address these issues this project set out to create a robotic simulator. The simulator in its current form provides useful functionality for the creation, testing and evaluation of robotic controllers. The sensor, objects, and probabilistic mapping that has been implemented provides a good base of functionality. There is much scope for further development of the simulator in multiple possible directions.

8. REFERENCES

[1] KUKA Robotics Corporation, Robotic Simulation revision 1.1, 2006

[2] M. Tim, IBM Developer Works - Open Source Robotics toolkits, IBM, 2006. http://www.ibm.com/developerworks/linux/library/l-robotools/.

[3] Francis Bernard, A Short History of CITIA and Dassault Systems, Dassault Systems 2003.

[4] Siemens, Press Release, 7th of May 2007, http://www.plm.automation.siemens.com/en_us/about_us/newsroom/press/press_release.cfm?Component=34092&ComponentTemplate=822.

[5] CyberBotics**,** About, 2011. http://www.cyberbotics.com/about.

[6] M. Somby, Updated review of robotics software platforms, linuxfordevices, 2008. http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Updated-review-of-robotics-software-platforms/.

[7] Player Project, homepage, 2010. http://playerstage.sourceforge.net/.

[8] Differentially driven robot, image taken from http://www.acroname.com/example/10013/wc-1a.jpg

[9] R. Siegwart, I.R. Nourbaksh, Introduction toAutonomous Mobile Robotics, 2004, ISBN 026219502X

[10] SICK LIDAR Sensor, image taken from www.mysick.com

[11] R. Sim, J.J. Little, Autonomous vision-based robotic exploration and mapping using hybrid maps and particle filters, Image and Vision Computing 2009, Vol. 27, ISSN 0262-8856.

[12] SciPy, 2011, www.scipy.org

[13] Robotic Swarm, Image taken from http://lh6.ggpht.com/_S1Gu2hX9S6c/SlUujpJy_CI/AAAAAAAALl4/DMsBQ_tXSl0/whole-swarm-from-above.jpg.