From Simulation to Real Robots with Predictable Results: Methods and Examples

S. Balakirsky, S. Carpin, G. Dimitoglou, and B. Balaguer

Abstract

From a theoretical perspective, one may easily argue (as we will in this chapter) that simulation accelerates the algorithm development cycle. However, in practice many in the robotics development community share the sentiment that "Simulation is doomed to succeed" [Brooks] p. 209. This comes in large part from the fact that many simulation systems are brittle; they do a fair-to-good job of simulating the expected, and fail to simulate the unexpected. It is the authors' belief that a simulation system is only as good as its models, and that deficiencies in these models lead to the majority of these failures. This chapter will attempt to address these deficiencies by presenting a systematic methodology with examples for the development of both simulated mobility models and sensor models for use with one of today's leading simulation engines. Techniques for using simulation for algorithm development leading to real-robot implementation will be presented, as well as opportunities for involvement in international robotics competitions based on these techniques.

Keywords

Simulation, performance metrics, algorithm development, mobility models, sensor models, robotics competition

1 Introduction

Using modeling and simulation to develop algorithms for motion planning provides development flexibility and extensive testing capability of the algorithm under a variety of operational environments and robot configurations.

The success of an algorithm depends upon, and is evaluated by, the robot's behavior upon execution. At the same time, as described in Kyriacou et al [Kyri08], robot behavior is affected by the robot's hardware, the implemented algorithm and the operating environment. The combination of these three elements results in the execution of a scenario in a complex, highly variable and often nonlinear system. Therefore, developing a general use simulator is a difficult task. Generality and simulation fidelity counterbalance each other in a simulation development.

During algorithm development for non-simulation based projects, the developer's considerations are somewhat bounded by: (i) the algorithm's input and output, (ii) the computation paradigm to be followed (e.g. divide-and-conquer, greedy, dynamic programming etc.) and (iii) any data structures that may be used to handle the data. The algorithm developer has minimal or no expectations about fea-

tures or extra feedback provided by the execution environment and no concern about the quality, fidelity or accuracy of the execution environment. Robotic algorithms tend to be developed for a specific task. Most fail at first as the developer makes relaxing assumptions about the robot hardware and the operating environment.

In algorithm development for simulation-based projects, the developer's considerations are significantly more expanded and include concerns related to the simulation platform. Issues such as accuracy, fidelity, determinism and overall realism have a direct effect on the computation and results.

1.1 Methodology for Algorithm Development

Currently, there is a void in the area of a formal methodology for simulation-based algorithm development. Such methodology should be able to treat the robot's hardware, its behavior and the operating environment as abstractions that can be independently -and jointly- specified and implemented. This would provide modularity and implementation-independent specification, which in turn enhance portability and allow model reuse. It would also enable the application of formal methods and automated model-checking. Most of the interactions between robots and the environment are developed based on a process of trial-and-error experimentation. Therefore, the overall accuracy of a robot simulator depends on the fidelity of its three components: the robot's hardware model (including both the base platform and the sensors), the algorithm, and the operating environment. Both the robot's hardware model and the operating environment are designed based on an estimation of how they should behave and interact.

Unfortunately, this results in environments and interactions based on assumptions that generate “virtual realities” that do not necessarily correspond to the physical environments and actual hardware. The assumptions may relax or extenuate one or more aspects of the simulation components (terrain, sensors and robot) and have a direct impact on the behavior of the developed algorithm. For example, a simplified terrain model or underlying physics engine with underestimated friction parameters would allow the algorithm to “drive” the robot too fast. This would result in miscalculating centrifugal forces when making turns, and inaccurate and unrealistic trajectories. Another example would be having a lower fidelity sensor payload that would be susceptible to a noisy environment resulting in failure to correctly interpret landmarks or signals. One final example would be using an omnidirectional holonomic steering when the physical platform is expected to be an Ackerman-steered vehicle. These are examples of simplifying assumptions or omissions that compromise the faithfulness of the simulation platform and significantly impact the effectiveness and appropriateness of a developed algorithm.

This chapter strives to address these issues by presenting techniques for validating the robot models and algorithms that are used in the simulation. While the

techniques are designed to be general purpose, specific examples are provided that relate how these techniques were applied to models that were developed for the Unified System for Automation and Robot Simulation (USARSim) [Usar09].

1.2 A Brief History of USARSim

The first release of USARSim was built by creating modifications to Epic's Unreal Engine 2 game engine[1]. It supported models for a few differential drive robotic platforms, a restricted set of sensors, and a small set of USAR specific test arenas. In addition, the robotic platforms could only be controlled through the use of the RETSINA [Syca98] multi-agent system software.

In 2005, USARSim was selected as part of the base infrastructure for the RoboCup Rescue Virtual Robot Competition. The virtual robot competition is an annual international event that highlights research in diverse areas such as multi-agent cooperation, communications networks, advanced mobility, mapping, and victim search strategies. In addition to the competition, USARSim management was taken over by the National Institute of Standards and Technology (NIST) and an international development community was established on the open source sourceforge.net website. While much of the original structure of the code was maintained, the code was reorganized and interfaces were standardized around SI units. The first official release (Version 1.0) was produced in October 2005.

A large-scale development effort accompanied the transition to sourceforge and the involvement of the RoboCup community. Version 3.31, released in July 2008 offers 15 different sensors, from odometry to an omnidirectional camera. 23 different robotic platforms are now available; these include wheeled robots, cars, tracked vehicles and flying robots. In addition, several of the sensors and robots have undergone rigorous validation of the forms outlined in this chapter in order to prove their similarities and difference from the real devices [BC08, Pepp07, Tayl07]. More information may be found at the USARSim website located at [Usar09].

The remainder of this chapter will detail these validation methods. First a look at robot platform validation will be conducted. This will be followed by a study of sensor validation. Next, algorithm development based on these validated models will be presented. Finally, a look at competitions that strive to utilize the simulated development cycle will be presented.

[1] Certain commercial software and tools are identified in this paper in order to explain our research. Such identification does not imply recommendation or endorsement by the authors, nor does it imply that the software tools identified are necessarily the best available for the purpose.

2 Robot Platform Validation

Robot platform validation is crucial to providing an accurate simulated model of a robotic system. A large body of work exists on modeling vehicle subsystems and the subsystem interactions. These simulations are capable of producing very accurate representations of the dynamics of mobile platforms and have been used in the design of commercial automotive systems. However, these systems are not capable of running in real-time on today's generation of low-cost desktop hardware. Since the objective of this validation is to verify that the simulated platform has *similar* capabilities as the physical hardware, we are able to trade-off simulation fidelity for real-time performance. This trade-off usually precludes modeling each sub-assembly of the robot, and the robot body is modeled as a single unit.

Our definition of *similar* performance is that *gross* platform behavior in both the physical and simulated platforms is verified. For example, if a simulated robot encounters terrain that would cause the physical platform to roll over, then the simulated platform should roll over as well. However, we find it acceptable for the physical and simulated platforms to experience different frequencies of vibration while traveling at the same speed over similar terrain.

In order to determine a more precise definition of what *gross* platform behaviors must be modeled, it is desirable to focus the validation on the domains that one would expect the robot model to encounter. In our case, this includes cluttered, uneven terrains with the robot performing various driving maneuvers. This has led to focusing the validation tests on capabilities such as platform maneuverability (acceleration, maximum velocity, turning radius), and rough terrain handling (center of gravity, climbing ability).

Figure 1: Example test method from test suite. The figure on the left shows the physical method while the figure on the right depicts the simulated version.

Our validation approach then became one of comparing the performance of the physical system to the simulated system in various relevant scenarios. This led to the development of a test-suite that allows objective comparisons to be made between the physical hardware and simulated models. However, it is often the case

that the individual who designed or owns the platform that is to be modeled is not the same as the modeler. In addition, the robot owner may object to having to send their platforms to an outside test facility where potentially untrained individuals would perform the testing. Therefore, a critical design criterion for the validation test suite was ease of test construction and administration. It was desired that the entire physical test suite could be built out of readily available, low-cost materials in a matter of a few days and that the tests themselves could also be conducted in a short amount of time.

One such test suite has been under development at the NIST and is in the process of becoming an international standard [Mess07]. A sample test method from the test suite (the "step test") is shown in the left hand side of Figure 1. The right hand side of Figure 1 shows the simulated version of this test. In the design of the simulated test methods, particular attention has been paid to validating that the test methods are accurately reproduced [Pepp07].

The tests that comprise the test suite may be decomposed (in order of increasing complexity) into the categories of static characteristics, hardware limits, hardware performance characterization, and interface performance characterization. The first two tests listed are the simplest to validate, and may usually be validated from design drawings of the physical platform. The static characteristics test verifies that the simulated model conforms to the same physical specification as the physical hardware. Robot dimensions, sensor placements, and actuator locations are verified. The true center of gravity (COG) of the robot is also established. This may be accomplished by either finding the balance point of the vehicle, or by using a set of scales to measure the force applied by each surface of the robot where ground contact is made. For the case of USARSim, the COG is an input parameter to the physics engine and no further tuning is required.

The hardware limits test verifies that the simulated model is constructed correctly. The range of motion of all moving parts is measured, and any motion constraints dictated by hard stoppages are verified. For example, a robot arm's range of motion may be reduced due to arm-body collisions. This test will verify that the simulated collision boxes are configured correctly so as to prevent the arm from traveling through the simulated robot body.

Hardware performance characterization is designed to verify the dynamic characteristics of the robotic system. As previous stated, it is not expected that a real-time physics engine will be capable of completely modeling the dynamics of a complex system such as a robot. However, it is important that the physics engine be "close enough" to modeling this system such that algorithms developed under simulation will be valid on the actual hardware. The hardware performance characterization consists of two separate tests. The difficulty of both of these tests is adjustable, and the tests are started from an "easy" configuration, with difficulty increasing until the platform is no longer able to accomplish the test (if possible).

Figure 2: Dash-test in flat (left) and 30 degree (right) configurations. This test is used to measure the platform's acceleration, deceleration, and maximum velocity. The P3 robot's laser scanner (located in the near field for the flat configuration and at the top of the ramp for the 30 degree configuration) is used to measure the distance to the robot under test as it approaches. A flat target is attached to the robot under test in order to maximize the number of laser beams that hit the target and provide noise averaging.

Figure 2 depicts the dash test that is designed to measure the acceleration profile, deceleration profile, and maximum velocity for a robot under the added stress of accelerating on an inclined plain. Once the maximum slope that a platform can climb is determined, the platform is tested at 100%, 50%, and 10% of the maximum slope as well as flat conditions. Certain platforms will flip over before reaching a slope that they lack the motor torque to climb. For these cases, a maximum slope is chosen based on platform and personnel safety.

Starting from zero velocity, the robot drives at maximum acceleration until full speed is reached. After a period of steady state velocity, the robot decelerates back down to zero velocity. During this test, a range sensor is used to observe the progress of a target fixed to the robot as the robot drives through the metric. In the case of Figure 2, a SICK LMS 200 that is mounted on a spare platform was utilized.

By measuring the difference in distance provided by successive hits on the target by a laser beam, a velocity measure is computed. For this metric, the velocity computation is performed at 10 Hz for each beam that impacts the target. All of the measured velocities are averaged together to provide a velocity estimate for a given run at a particular time instant.

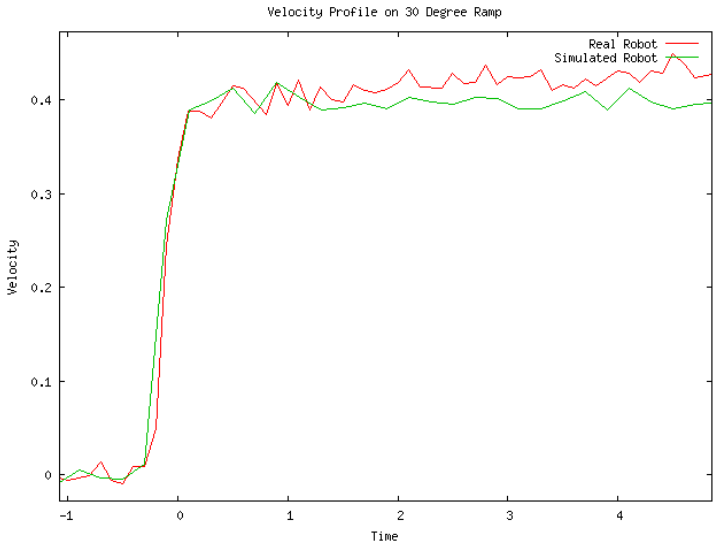

Figure 3: Velocity profiles for both a physical model and its simulated counterpart while operating at a 30 degree slope. The plot depicts time(s) vs. velocity (m/s).

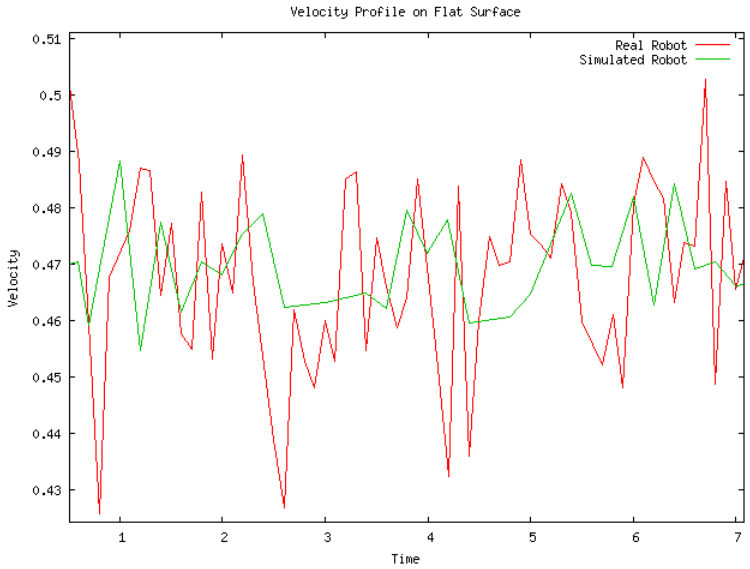

Figure 4: Velocity profiles for both a physical model and its simulated counterpart while operating on flat ground. The plot depicts time(s) vs. velocity (m/s).

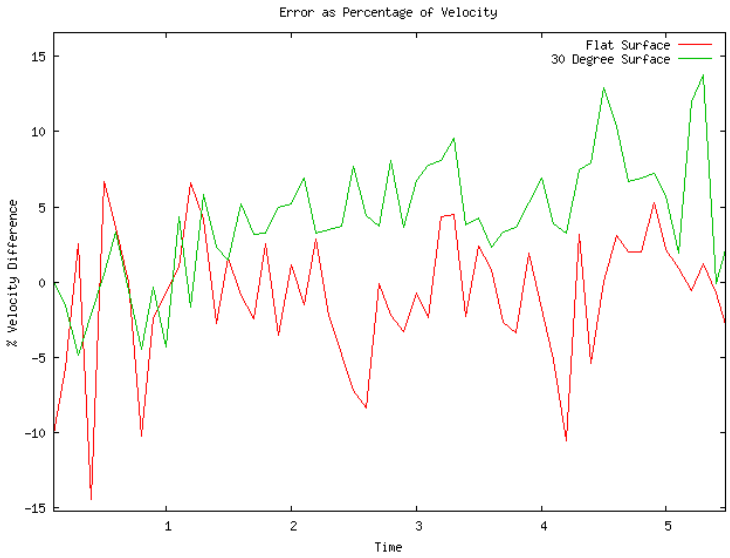

Figure 5: This figure show the velocity error for the simulated system as a percentage of the desired velocity. Plots for both the 30 degree ramp and flat test methods are shown.

This test is performed 10 times for each slope setting. The results from the individual tests are then weighted based on the number of beams reporting velocities, and averaged together to form a velocity profile.

Figure 3 and Figure 4 show the measured profile for a platform under test compared to a tuned model from simulation. The test for Figure 3 was performed on the 30 degree slope ramp while the test shown in Figure 4 was performed on a flat surface. In order to achieve this tuning, the maximum track velocity and motor torque settings of the simulated model are tuned. The exact technique used for this tuning procedure is of course simulation system dependent. For USARSim, the physics engine has the ability to dynamically modify the model's parameters. The authors took advantage of this capability to dynamically modify parameters during a run until the average of actual velocities matched our desired average velocities. These modifications are currently carried out by hand; in the future it is desired that this tuning will be automated. Figure 5 depicts the velocity error between the simulated platform and the real platform as a percentage of the real platform's velocity. As may be seen in this figure, the model is able to consistently capture the velocity profile for the flat test area. However, there is an acceleration component that the real platform exhibits when traveling up a steep hill that is not captured by the simulation. It may be seen that there is increasing error with time as the real platform accelerates up the hill. This error has an upper bound of about 15 %.

The second hardware performance test utilizes the metric shown in Figure 1. This test measures the ability of a platform to climb steps of various heights. The test is started with a step of height 5cm and the step height is increased by 5cm increments until the platform is unable to reach the landing. The tubes on the leading edge of the step are free-spinning and are designed to prevent the platform from gripping the step and pulling itself up. Video of the physical platform performing this test is collected, and the simulation is tuned so that the platform model achieves similar performance. The main simulation parameters that are adjusted during this process deal with tire properties. However, it may be necessary to tune the COG and motor torques as well.

The final area for platform validation concerns the control interface to the platform. It is desired that when presented with identical command streams, the physical and simulated platforms will exhibit similar responses. For hardware platforms that support a USARSim supported interface, this test is a simple matter of applying the same command stream to both the physical platform and the simulated platform. The platforms' trajectories may then be measured and compared. Such a test was first performed by [BC07]. Differences in behavior may usually be compensated for by scale factors on the command stream. When the interfaces do not support identical command streams, but a command API is available for the platform under test, a command translator may be constructed. The test is then able to proceed as described above. In the worst case, the command syntax is proprietary, and the only way to discern what input is being delivered to the physical platform is by measuring battery current to the motors using a system such as the one described in [SWRI].

Once all of the above tests have been performed and the platform model is complete, it is still necessary to validate the validation. This is accomplished by measuring the physical and simulated performance of the platform on a subset of the NIST test methods that were not used for the initial validation; for example, staircase climbing. If performance of the two platforms on these methods is similar, then one may consider the platform to have been successfully validated.

3 Sensor Validation

Sensor validation follows the same general-purpose validation methodology that is used to validate other components of the simulation system, e.g. robots and actuators. The tenet is to implement the same experiment in simulation and in reality, and to numerically compare the results. The choice of the numeric metric to use for this comparison is in general non trivial, and potentially amounts to a demanding research question on its own. In certain cases it will be possible to directly compare the outputs of simulated and real sensors, while in other scenarios it will be preferable to contrast the results of simple computational procedures that process the sensor output. The appropriate choice depends on the sensor under

consideration, and no one-catch-all formula exists, even though some indications can be extracted from the examples presented in the following.

Setting up a virtual-reality experiment implies various programming and modeling tasks. Clearly, an appropriate sensing model needs to be incorporated within the simulation. Additionally, a faithful model of the real environment has to be developed into the testing environment. This is an unavoidable step because the surrounding environment naturally influences the sensing process. The goal of the sensor validation stage is twofold. Not only is it important to determine the degree to which a simulated sensor delivers results that can be extrapolated to infer results of the real world system, but it is also pivotal to identify the sensors that fail to replicate phenomena observed with their real counterparts. In the past, a set of very different sensors have been modeled and validated [CWL06, CSN06, BC08], and we provide a short synopsis of the methodology and results.

According to our experience, and not surprisingly, the best way to develop a realistic sensor model is to embed, directly into the simulation engine, a computational method that replicates the physical phenomena occurring in the real device. This approach necessarily implies the inclusion of appropriate models of the noise affecting the process. This last step is particularly challenging because these noise sources are often difficult to characterize or add to the system. In the following we illustrate the methodology and findings we obtained while modeling and validating two different sensors, namely a laser range finder and a GPS sensor. Two reasons motivate the choice of these sensors as working examples to demonstrate our general-purpose validation method. First, these are among the most popular sensors used to solve the localization problem indoor and outdoor, respectively. Secondly, it is rather straightforward to implement their underlying working principles within the simulation environment by using ray-tracing primitives offered by the game engine.

3.1 Laser range finder

Laser range finders find applications in a variety of robotic tasks, such as navigation and map building, just to name a few. These devices are essentially time of flight devices, i.e. they measure the time elapsed between the moment a beam is emitted and its reflection comes back to the sensor. By relating this time to the physical properties of the signal being sent and sensed, the distance between the sensor and the surface that reflected the beam can be inferred. These sensors usually emit a series of beams covering a wide angular range. The Sick PLS, probably the most popular sensor of this type, covers a range of 180 degrees with beams spaced either 1 or 0.5 degrees apart. Such a sensor is straightforwardly mimicked in USARSim by exploiting the ray-tracing primitive operation made available by the underlying game engine. Unfortunately, ray-tracing can be considered an error-free function in the simulated environment that does not take into account the beam flight time or the surface of reflection – important error sources in real laser range finders. Therefore, appropriate noise needs to be added to the simula-

tion in order to replicate real sensor behavior. Two noise sources mainly affect this sensor. The first are jitters observed in the measured distances due to errors while measuring time intervals. This error is fairly simple to model and imitate in the simulation environment. The second error stems from unreturned beams, i.e. beams that are not reflected by the surface they hit. This second error is instead rather difficult to replicate in simulation because it would imply labeling all the surfaces in the simulated world as reflective or non-reflective, or assigning a reflective coefficient to each surface. While this can be achieved in principle, it implies an amount of work that is definitely not worth the effort and has therefore not been implemented in the simulator.

Since the sensor under consideration returns numerous values, it is possible to compare real and simulated data point by point, or to relate the result of a function computed on the sensed data. This latter approach is embraced in this case study. In particular, we will show the results of the well-known Hough algorithm for line detection. This example is particularly appealing because it illustrates the advantage of simulation from two points of view. Firstly, the algorithm requires hand tuning of a few parameters. This process is driven by the data being processed and may be particularly time consuming if being performed with a real device. Secondly, the algorithm is known to be fairly sensitive to noise in the data. By altering the amount of noise affecting the simulated sensor it is therefore possible to assess the robustness of the algorithm to different noise levels. This later aspect is in our view one of the most appealing aspects of the simulation system we propose. Obviously, this assessment will be appropriate only if it will be possible to show a strong correlation between simulation and the real world system.

Figure 6: Real and simulated robots observed while validating the range scanner simulation. The cross in Figure 7 shows the position of the robot in the environment, where these pictures were taken.

Figure 6 shows how the experiment is executed, i.e. the simulated and the real robot are positioned at the same place in the physical and simulated environment.

Figure 7 shows a superimposition of the data collected by the two sensors while in the position depicted in Figure 6. It is evident that the data returned by the two sensors are very close to each other. This visual evidence can be numerically assessed by comparing how the Hough transforms computed from the two datasets relate to each other. It should be stressed that this choice is not the only possible one, nor necessarily the best one, but was rather selected because the simulation environment was being used to perfect a robot control system where line detection via the Hough transform was a building block for the higher layers.

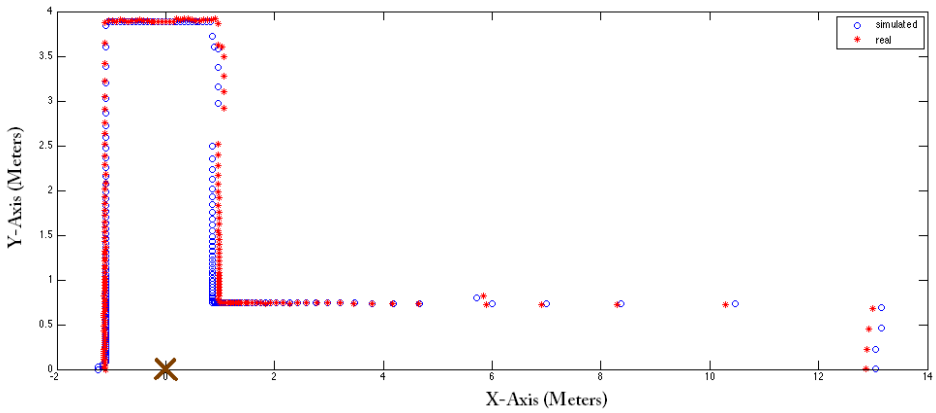

Figure 7: Data collected by the simulated and real sensors. The location of the robot, from which the data was taken, is marked by a cross (at the 0,0 coordinate). The cross also shows where the pictures in Figure 4 were taken.

Without getting into a technical discussion of how the Hough transform is computed or used later on (the reader is referred to [CWL06] for more details), we just mention that its final result is stored in a grid of non negative integer values. After computing the Hough transform for both datasets over a grid with 2592 cells, the average difference between values stored in corresponding cells is 0.0328. In order to put this value into context, it should be mentioned that the highest value stored in the grids is 6, and that for more than 99% of the cells, the difference is smaller or equal than 1. These values support the claim that results obtained with the simulated sensor can be extrapolated to the real sensor as well.

3.2 Global Positioning System

Robots performing in outdoor environments rely more and more often on the availability of a Global Positioning System (GPS) sensor to obtain an estimate of their global position. The GPS system depends on the presence of a set of satellites orbiting around the earth according to known trajectories. In essence, a GPS

sensor periodically receives signals from the orbiting satellites and uses this information to infer its position. A signal emitted by a satellite is received by a GPS receiver only if there is line of sight between the two. The GPS receiver then processes all information included in the received signals and computes its position. The more satellites that are within line of sight, the more accurate is the pose estimation produced by the GPS sensor (the reader is referred to [BC08] for a more detailed description of the computational aspects and additional results).

Based on this working principle, a GPS sensor simulator can be developed as follows. Every time a new reading from the GPS sensor is requested, ray tracing is performed between the simulated receiver and all of the satellites in order to determine which ones are visible from the receiver. The receiver's position is then computed through simple mathematical computations. Gaussian noise is superimposed to the computed position based on the number of detected satellites. This brief description reveals that the pose computed by the simulated sensor replicates the computational approach exploited by the real receiver. In particular, tracing between the receiver and the satellites entails two different kinds of knowledge. First, at any point in time it is necessary to know the position of the satellites. Secondly, while performing the tracing, it is necessary to take into account the presence of obstacles possibly obstructing the path between the receiver and a satellite.

Since the accuracy of the pose estimation is intimately related to the number of detected satellites, and this value is commonly made available by most GPS receivers, one good numeric value that can be used to compare the performance of the real and simulated device is the number of satellites detected. To do so, it is not only necessary to know the satellites' positions, but also to include into the simulated environment all the elements that may possibly preclude the detection of a satellite. In the real world experiment, a mobile robot equipped with a GPS receiver moves in the University of California, Merced campus between a set of multi-story buildings. Hence, in the simulated experiment, corresponding geometrical models of these buildings were inserted in the test environment. Given that the number of detected satellites influences the accuracy of the estimated pose, the other numerical parameter that can be numerically compared is the difference between the two position estimates. Obviously, this value should be as close to 0 as possible. For the GPS sensor, it is therefore possible to numerically compare the values produced by the sensor under consideration.

It should be mentioned that satellite occlusion is definitely not the only source of uncertainty for GPS sensors, but it is surely the easiest to faithfully model. Moreover, as evidenced in the experiments described in the following, while just including this source of error, a satisfactory performance is observed. In this context *satisfactory* means that the numerical values observed in the two experiments are close to each other thus providing evidence of the accuracy of the simulation.

Figure 8: Comparison between the traces produced by the real and simulated GPS sensor. The traces have been superimposed to a satellite image of the University of California, Merced campus.

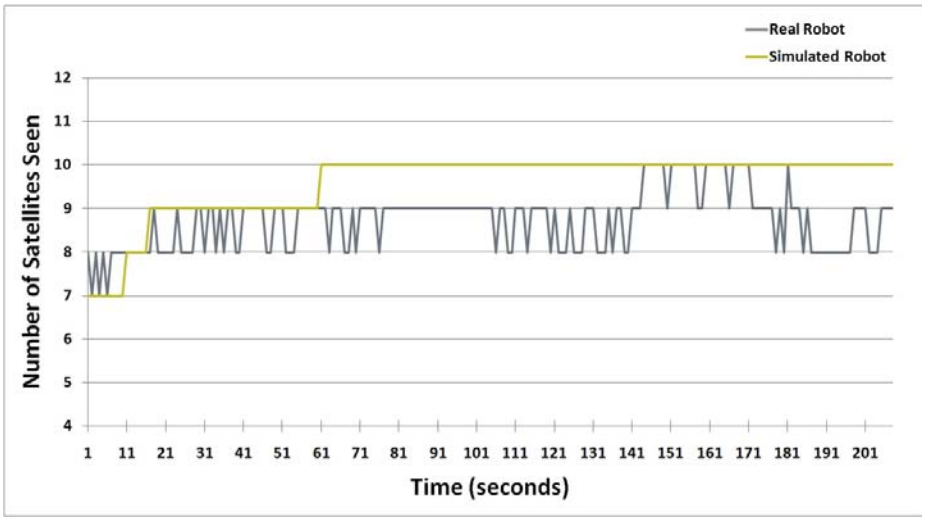

Figure 9: Number of detected satellites as a function of time for the yellow experiment in Figure 8.

The first experiment is illustrated in Figure 8. Both the simulated and the real robots were driven though three paths in the campus quad. The picture shows the pose returned by the simulated and real GPS sensors. It is evident, by visual inspection, that the two paths are similar. However, this visual correspondence can be numerically evaluated as well. In fact, the average difference between the two pats is 1.49 meters for the first path (yellow), 4.57 meters for the second (blue), and 2.37 meters for the third (green). Even though these numbers might seem high, the reader keep in mind that the typical accuracy of civil GPS sensors is 3 meters. The second experiment is illustrated in Figure 9.

The chart shows the number of satellites seen by the real and simulated robot. In this case it is manifest that the simulated sensor almost always sees a number of satellites greater than the real one. This is easily explained by the fact that the simulated model does not include the numerous trees that are present in the real world. In any case, as evidenced in the first experiment, it turns out there is a good agreement between the data returned by the two sensors, therefore one can be confident that results obtained in simulation can be extrapolated to real world systems.

4 Algorithm Development

Robot simulation platforms provide versatile environments for the development of robotic algorithms. Developers can easily test algorithm execution using multiple robotic platforms in various synthetic environments and operational scenarios. This makes simulation-based algorithm development an indispensable research tool that allows extensive testing of robotic behavior in configurations and environments in which physical testing would otherwise be prohibitively expensive or even impossible to achieve.

Here, we focus simulation-based algorithm development analysis in the context of mobility and robot motion planning. Motion planning is an active research area and has been studied for decades. Depending on the approach, it has been referred to as motion planning, path planning or trajectory planning, but in essence it has always referred to the same class of problems. The seemingly trivial task of instructing a robot to move between two points on a plane via a collision-free path often becomes surprisingly difficult to generalize for any two points on any plane with any configuration space.

Early formulations of the problem appeared almost forty years ago using graphs for shortest path searches (visibility graphs) between polygonal obstacles [Nils69]. Another formulation of the motion planning problem, known as the *piano movers' problem* [Cann89, Schw83, Schw83a] was studied from a geometry perspective. Not unlike similar current problems of robots navigating past obstacles, the examined task was to successfully move a piano between house rooms while avoiding obstructions. The examination ignored any kinematic constraints such as how piano wheels -if they existed- would enable or constrain steering. It

also disregarded speed and any temporal aspect such as elapsed time, related to the navigation of the piano from room to room.

Since this first formulation, many different motion planning approaches and techniques have been developed such as grid sampling, configuration spaces [Loza80], roadmaps [Cann88], randomized potential fields [Barr97], probabilistic roadmaps (PRMs) [Kavr96], rapidly-exploring Random Trees (RRTs) [LaVa00] and many others. Some of these techniques have crossed the boundaries of robotics research and have been applied in other domains and disciplines. For example, motion planning algorithms have been applied to biology, computer animation and medical surgery for protein folding pathways [Song01, Thoma05], virtual reality [Shen95] and laparoscopy procedures [Fara00], respectively.

Today the formulation of motion and mobility problems is significantly more complex. Task completion is extended with additional requirements such as maximizing speed, minimizing navigation distance (e.g. shortest-path) and energy consumption (i.e. battery power) or maximizing path accuracy. Compounding the complexity is: the operation in dynamic environments with uneven terrains and moving obstacles, the utilization of sensors and their imperfect readings and even the coordination and interaction of multiple robots in the same area.

The volume, breadth and depth of research and multi-disciplinary application of motion planning techniques emphasize the importance of the work in this area and yet it illustrates the inherent complexity of the problems to be solved. Even the original problem of moving a piano would be a discouraging proposition for anyone to physically validate using trial-and-error testing. Attempting to test, verify and validate new algorithms on complex robotic platforms operating in dynamic and even hostile environments, such as volcanoes, underwater, or on distant planets, is a daunting and prohibitively expensive, if not impossible task. This leads one to explore the utility of performing algorithm development in a safe, inexpensive simulated environment.

However, the effectiveness and appropriateness in simulation-based algorithm development is highly dependent on the characteristics of the simulation platform. The following are some desirable simulation platforms characteristics:

(a) *Generality*: to allow the testing and experimentation of different algorithms on different simulated robots with various sensor payloads and in different simulated operating environments.
(b) *Accuracy:* because simulated algorithm executions are expected to approximate reality. The narrower the gap between the simulation execution and the corresponding physical execution, the more accurate the simulation platform is.
(c) *Computational soundness:* which implies that executed algorithms and computations are deterministic. Non-deterministic simulations compromise the repeatability of the execution which is necessary to determine the stability of the simulation platform.

From a software engineering perspective, simulation platforms must also have features that support and augment the above characteristics:

(a) *Extensibility:*, is related to (i) *framework extensibility* which includes *scenario* and *environment extensibility* so that new robot profiles, terrain information, environment artifacts, configuration spaces, sensor types and other components can be easily added or updated; (ii) *algorithm development extensibility* so that existing algorithms can be extended, new algorithms can be developed or easily replaced in either case without affecting the underlying framework.
(b) *Temporal contro*l: to allow real-time display and simulation speed and replay capability.
(c) *Realism:* in rendering the visual aspects of the simulation so robots do not seem to navigate over obstacles. Realism also applies in the rendering of sensor readings that may be inaccurate due to inherent noise in the environment.
(d) *Interface versatility and richness: to* enable easy control of the simulator and experiment design while providing a high degree of configure-ability and feedback during simulation executions.
(e) *Logging and meta-information collection mechanisms:* to provide a way to collect and analyze quantitative measurements from executed experiments.
(f) *Scalability:* to allow the inclusion of multiple robots or to allow framework extensibility as discussed earlier.

4.1 Criticisms and Advantages

Regardless of the particular characteristics and features of a simulation platform, the inherent complexities of robotic environments may introduce significant differences between the simulated and the real environment. The combination of extensive robot configuration parameters, diverse terrain options, high variability in sensor readings and error rates, coupled with operational scenarios that change in real-time, create very dynamic environments that are very hard to accurately replicate in a simulation. All of these negatively affect the reliability of simulation platforms and the algorithms developed using them. As mentioned earlier, further aggravating the situation is the lack of a formal methodology for simulation-based algorithm development.

Despite these criticisms, simulation-based algorithm development has advantages over the alternative, the direct development on the physical platform and environment. Simulations allow for:

- Inexpensive and rapid setup of a virtual experimentation environment;
- Time execution acceleration, providing the developer insight on how an algorithm would execute after long time periods without having to execute them in real time;
- Consistent environment for experimentation;
- Test and experiment repetition;
- Easy examination of incremental development progress;

- Risk reduction for damage to hardware or the operating environment due to errors in execution.

Many simulation environments and frameworks have been developed and can be used to model different environments and simulate tasks. Some are described as general frameworks. Others specialize and focus on certain aspects of simulation. Specialized frameworks tend to provide greater accuracy and fidelity for the particular task they model, but lack extensibility which general frameworks tend to be better equipped to provide. For example, a non-comprehensive list of robotic simulation platforms and frameworks includes: Darwin2K [Lege00], Gazebo [Koen04], Player/Stage [Gerk03], SimRobot [Laue06] and Webots [Michl98].

In the context of a Virtual Manufacturing Automation Competition, we embarked on the simulation-based development of two algorithms: one to address waypoint navigation and another to perform docking. Both of the algorithms were developed, exercised, tested and verified using a simulation infrastructure comprised of USARSim [Balak06, Carp07], a Gamebots [Adob01] variant of the Unreal Engine [Unre08] and the Mobility Open Architecture Simulation and Tools (MOAST) framework [Bala08].

The USARSim infrastructure supports algorithm experimentation. The MOAST framework allows the separation of algorithm development from the different system levels (echelons) and configurations which control different aspects of the experiment.

The operational scenario of the navigation algorithm was based on waypoints for terrestrial navigation. Waypoints are longitude and latitude-based coordinate locations used to construct routes. In the context of robotic navigation, waypoints can serve as the guides for a robot to follow a specific path. While following waypoints is trivial, idiosyncrasies of the robotic platform and the operating environment may render this type of navigation inaccurate and ineffectual under some circumstances. An example of such a case is with Ackerman-steered vehicles. The limitation of such vehicles is their inherent inability to perform sharp turns to reach waypoints in very close proximity. Specifically, waypoints located within the vehicle's instantaneous center of rotation (ICR). This case presents two navigation challenges. First, it may cause the vehicle to deviate considerably from the prescribed path. Second, it may cause the vehicle to enter an endless spiral path attempting to reach the practically unreachable waypoint. To address these challenges we developed an algorithm that visits all waypoints on an arbitrary projected path, minimizing path deviation, avoiding spiral movements and continuing navigation on the projected path. Using an arbitrary navigation path of multiple segments, the algorithms were executed using a simulated Ackerman-steered, automated guided vehicle (AGV) to navigate the path. The navigation path or the "world," along with an arc file, were provided as input. The simulated vehicle generated the waypoints to be followed to complete the path from the initial location to a goal waypoint. The simulation environment provided a number of confi-

gurable parameters for the vehicle that could be adjusted to modify the navigation performance.

Table 1 is an example of the default, configurable navigation parameters.

Table 1. Unit Loader Default Navigation Parameters.

Vehicle (Unit Loader) Parameters	
MAX_TRAN_VEL	5
MAX_TRAN_ACC	50
MAX_ROT_VEL	600
MAX_ROT_ACC	6000
V_CUTOFF_ANGLE	210
W_CUTOFF_ANGLE	10
CONTROL_POINT	0.3 0 0 0 0 0

For the experiment, a Unit Loader AGV was used, and all parameters but velocity were kept at their default settings.

In one experiment, the focus was on path navigation accuracy and speed of path completion. To examine the range of the results, six simulation executions were performed with different vehicle velocity parameters (Table 2).

Table 2: Algorithm execution results with variable velocities.

Execution ID	*Velocity (m/sec)*	*Time (sec)*	*Number of Reverse Actions*
001	1.0	443	1
002	2.0	235	1
003	2.5	215	7
004	3.0	220	11
005	4.0	182	16
006	5.0	173	20

By examining the results and plotting the velocity with respect to the time it took to complete the path, it is shown that as velocity increases, the time required to complete the path decreased (Figure 10).

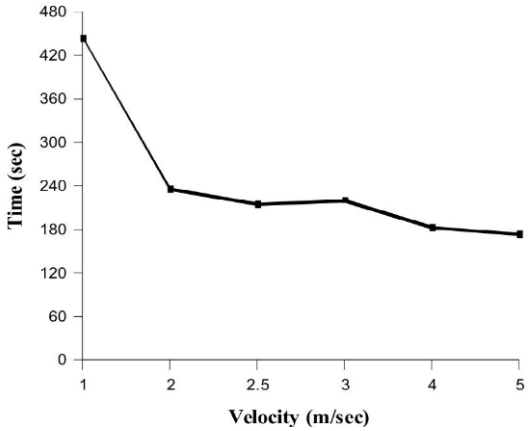

Figure 10: Plot of the time over variable velocity of multiple executions o the same navigation path.

Intuitively, this is an expected result; however, it does not reveal any information about the accuracy of the followed path. In fact, it hides the effect that the higher velocities have on accuracy in following the path. Figure 11 reveals that while the time for path completion is reduced with higher speeds, the amount of corrective actions, in this case, the number of times the vehicle must reverse to continue reaching all of the path waypoints also increases dramatically.

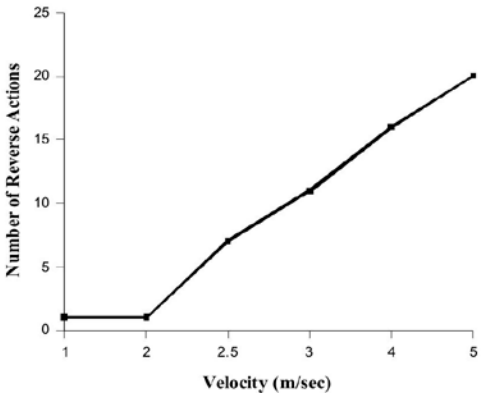

Figure 11: Plot of the number of reverse actions over variable velocity of multiple executions of the same navigation path.

For velocities that do not exceed 2.0 m/sec the number of reverse corrective actions is the same and negligible. However, even a 0.5 m/sec velocity increase causes the number of corrective actions to increase by a factor of seven. Doubling the velocity from 2.0 m/sec to 4.0 m/sec results in a sixteen-fold increase of the

number of corrective actions. The steep increases in the number of corrective actions indicate that the vehicle must maneuver, and therefore deviate from the original path to compensate for poor orientation from waypoints. These measurements and analysis was possible only because of the ease in tuning different parameters, afforded by the simulation platform.

The simulation environment allowed us to develop a successful navigation algorithm and to analyze the performance of the algorithms. To examine the range of the results, six simulation executions were performed with different vehicle velocity parameters. By examining the results and plotting the velocity with respect to the time it took to complete the path, we were able to identify additional elements about our algorithm. As expected, when velocity increased, the time required to complete the path decreased. At the same time, increased velocity impacted path accuracy. While the time for path completion reduced with higher speeds, the number of corrective actions, where the vehicle must reverse and change orientation to continue reaching all of the path waypoints, also dramatically increased. We were also able to identify velocity ranges where the number of corrective actions was negligible or very small, along with velocity ranges that caused a seven-fold, and on other occasions, sixteen-fold increase in the number of corrective actions.

Clearly, the simulation-based algorithm development gave us an advantage by being able to execute the algorithms multiple times and being able to collect quantifiable, measurable data related to the performance of the robotic platform. Attempting to perform the same development on an actual platform would have taken a tremendous amount of time and would have risked significant damage to the robot during the early stages of development.

The operational scenario of the second algorithm was based on properly docking a vehicle to a conveyor belt docking station. Autonomous robot docking requires accurate path following and accurate alignment with the target location, typically a docking station.

Using the simulator, we developed a novel, partially heuristic algorithm that allows accurate docking for Ackerman-steered vehicles. Multiple experiments were performed to better understand and analyze the heuristic element of the technique. The algorithm was exercised in gradually smaller rooms to evaluate the versatility of the algorithm under more constrained environments. The results underscored the impact of the vehicle's steering characteristics in docking precision and may prove valuable in attempting to remove the heuristic element in future algorithms.

Again the simulation-based algorithm development gave us an advantage by being able to execute the algorithms multiple times and in variable-sized rooms. Attempting to perform the same development on an actual platform would have: taken a tremendous amount of time, it would have been difficult to replicate the extensive testing in various sized rooms, and it would have risked significant damage to the robot during the early stages of development. Preliminary validation based on visual inspection of the algorithm on a physical platform (a NIST modified ATRV platform) demonstrated that the docking objective was achieved.

5 Competitions

The framework under which the above algorithm development took place was the IEEE Virtual Manufacturing Automation Competition (VMAC) [VMAC09]. Both this competition, and the RoboCup Rescue Virtual Robots Competition [ROBO09] utilize the USARSim simulator as part of their infrastructure.

The RoboCup Rescue Virtual Robots Competition falls under the umbrella of the RoboCup Rescue competitions. The RoboCup Rescue competitions provide a benchmark for evaluating robot platforms for their usability in disaster mitigation and are experiencing ever increasing popularity. Roughly speaking, the league vision can be paraphrased as the ability to deploy teams of robots that cooperatively explore a devastated area and locate victims. Farsighted goals include the capability to identity hazards, provide structural support and more. RoboCup Rescue is structured in two leagues, the Rescue Robot League and the Rescue Simulation League. Whereas the Rescue Robot League fosters the development of high-mobility platforms with adequate sensing capabilities, e.g. to identify human bodies under harsh conditions, the Rescue Simulation League promotes research in planning, learning, and information exchange in an inherently distributed rescue effort. The Rescue Simulation League contains three competitions; the Virtual Robot Competition, the Agent Competition, and the Infrastructure Competition. The Virtual Robots competition simulates, compared to the Rescue Agents competition, small teams of agents with realistic capabilities operating on a city block-sized scenario.

The Virtual Robot competition, first held during the RoboCup competitions in 2006, provides a realistic simulation environment for simulating conditions after a real disaster, such as an earthquake, a major fire, or a car wreck on a highway. Robots are simulated on the sensor and actuator level, making a transparent migration of code between real robots and their simulated counterparts possible. The simulation environment allows evaluation of the performance of large robot teams and their interactions. For example, whereas in the real robot competition there are usually only one or two robots deployed, in the Virtual Robot competition teams of up to twelve robots are deployed. Furthermore, the simulator provides accurate ground truth data allowing an objective evaluation of the robots' capabilities in terms of localization, exploration and navigation, e.g. avoidance of bumping. More information on the virtual rescue competition may be found in Balakirsky et al. [Bala07].

The VMAC competition focuses on Automated Guided Vehicles (AGVs). These vehicles represent an integral component of today's manufacturing processes. Major corporations use them on factory floors for jobs as diverse as intra-factory transport of goods between conveyors and assembly sections, parts and frame movements, and truck trailer loading/unloading.

The competition design was based on the successful RoboCup Rescue Virtual Robots Competitions. Since all code used in these competitions is open source,

participants are able to learn from their competitors and concentrate their research in their particular areas of expertise. It was envisioned that researchers from multi-agent cooperation, mapping, communications networks, and sensory processing backgrounds would all be interested in participating.

The initial competition design was formulated by using the SCORE framework [Schl06]. This framework specifies that an overall system scenario be defined, and then basic elemental skills that allow for the successful completion of the scenario be extracted. Systems are then evaluated on both their ability in the elemental tasks as well as the overall scenario.

From the outset, the competition was to be based on real-world scenarios. Based on NIST's industry outreach effort, the scenario chosen was a factory setting that had significant clutter, maze-like passageways of various widths, and dynamic obstacles. The objective was to have several Ackerman-steered AGVs pick-up packages at a central loading station, and deliver these packages to one of several unloading stations. The package destinations were encoded in a Radio Frequency IDentification (RFID) Tag on each package.

Utilizing the SCORE framework, this scenario was decomposed into elemental tasks that included traffic management, route planning, accurate path following, and docking with loading/unloading stations. While the baseline code provided to the teams was capable of performing the objectives, it was far from optimal.

For the first running of the competition, a decision was made to only compete two of the basic elemental tasks; accurate path following, and docking. One team's experiences with the virtual development cycle for this task are outlined in the previous section. More information on the VMAC may be found on the VMAC webpage [VMAC09] or in an overview by Balakirsky et al. [Bala08b].

6 Conclusion

Robot simulators are useful tools for developing algorithms to control robot behavior. However, the execution of robotic algorithms is not confined to the inner-workings of a CPU but have a physical manifestation in real environments with physical robots traversing real terrains and avoiding real obstacles. This means that robotic algorithms are assessed both theoretically and practically.

Simulation platforms allow the development of algorithms in a safe environment where execution errors are benign and hardware reliability is not an issue. Such platforms allow great flexibility in designing complex environments and testing algorithms repeatedly under multiple scenarios.

At the same time, these platforms have limitations. Algorithm development on a simulation assumes that the information about the environment is accurate. Yet the complexity of the operating environment can be daunting and certain variables such as terrain characteristics or sensor sensitivity may be omitted or ignored. In addition, all of the elements of a simulation: the robot, the terrain and the sensor payload are subject to simplifying assumptions. These assumptions may reduce

the realism of the simulation enough to render the developed algorithm insufficient for deployment in a real world environment.

Still, none of these concerns are enough to offset the tremendous benefits of simulation platforms in terms of cost savings, risk reduction by testing on the real platform, having a consistent experimentation environment and having the ability to repeatedly test and collect measurable, quantifiable data.

7 References

[Adob01] Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S., Kaminka, G.A., Schaffer, S., Sollitto, C. "GameBots: A 3D virtual world test bed for multiagent research," Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada, 2001.

[Bala06] Balakirsky, S., Scrapper, C., Carpin, S., Lewis, M. "USARSim: providing a framework for multi-robot performance evaluation," Proceedings of PerMIS, 2006.

[Bala07] Balakirsky, S., Scrapper, C., Carpin, S., Lewis, M., "USARSim: A RoboCup Virtual Urban Search and Rescue Competition," Proceedings of the 2007 SPIE Unmanned Systems Technology IX, Defense and Security Symposium, 2007.

[Bala08] Balakirsky, S., Proctor, F., Scrapper, C., and Kramer, T., "An Integrated Control and Simulation Environment for Mobile Robot Software Development," Proceedings of the ASME Computers and Information in Engineering Conference, 2008.

[Bala08b] Balakirsky, S., Madhavan, R., and Scrapper, C., "NIST/IEEE Virtual Manufacturing Automation Competition: From Earliest Beginnings to Future Directions," Proceedings of PerMIS, 2008.

[Barr97] Barraquand, J., Kavraki, L., Latombe, J., Motwani, R., Li, T., and Raghavan, P. 1997. "A random sampling scheme for path planning," Int. J. Rob. Res. 16, 6 (Dec. 1997), 759-774.

[BC07] Balaguer, B., Carpin, S., Balakirsky, S., "Towards Quantitative Comparisons of Robot Algorithms: Experiences with SLAM in Simulation and Real World Systems," Workshop on Performance Evaluation and Benchmarking for Intelligent Robots and Systems at IEEE/RSJ IROS, 2007.

[BC08] Balaguer, B., Carpin, S. "Where Am I? A Simulated GPS Sensor for Outdoor Robotic Applications," Proceedings of the First International Conference on Simulation, Modeling and Programming for Autonomous Robots, 2008, 222-233

[Brooks] Brooks, R., Matarić, M., "Real Robots, Real Learning Problems", in *Robot Learning,* Jonathan H. Connell and Sridhar Mahadevan, eds., Kluwer Academic Press, 1993, 193-213.

[Cann88] Canny, J. F. “The Complexity of Robot Motion Planning,” MIT Press, 1988.

[Cann89] Canny, J. 1989. On the “Piano Movers” series by Schwartz, Sharir, and Ariel-Sheffi. In the Robotics Review 1, O. Khatib, J. J. Craig, and T. Lozano-Pérez, Eds. MIT Press, Cambridge, MA, 33-40.

[Carp07] Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C. “USARSim: a robot simulator for research and education,” Proceedings of the IEEE 2007 International Conference on Robotics and Automation, 2007, 1400-1405.

[CSN06] Carpin, S., Stoyanov, T., Nevatia, Y., Lewis, M., Wang, J. “Quantitative Assessments of USARSim Accuracy,” Proceedings of PerMIS 2006.

[CWL06] Carpin, S., Wang, J., Lewis, M., Birk, A., Jacoff, A. “High fidelity tools for rescue robotics: results and perspectives,” Robocup 2005: Robot Soccer World Cup IX, LNAI Vol. 4020, Springer, 2006, pp. 301-311.

[Fara00] Ali Faraz a1, Sharam Payandeh, Kinematic modelling and trajectory planning for a tele-laparoscopic manipulating system, Robotica (2000), 18:4:347-360 Cambridge University Press

[Gerk03] Gerkey, B., Vaughan, R., Howard, A. “The player/stage project: Tools for multi-robot and distributed sensor systems.” Proceedings of the International Conference on Advanced Robotics (ICAR), 2003, pp.317-323.

[Kavr96] Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; Overmars, M. H., “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” IEEE Transactions on Robotics and Automation 12 (4), 1996, pp. 566–580.

[Koen04] Koenig, N., Howard, A. “Design and use paradigms for Gazebo, an open-source multi-robot simulator,” IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004, pp. 2149-2154.

[Kyri08] Kyriacou, T., Nehmzow, U., Iglesias, R., Billings, S. A. “Accurate robot simulation through system identification”, Robot. Auton. Syst. 56, 12, 2008, 1082-1093.

[Laue06] Laue, T., Spiess, K., Röfer, T. (2006). “SimRobot - A General Physical Robot Simulator and Its Application in RoboCup,” RoboCup 2005: Robot Soccer World Cup IX, No. 4020, 2006, pp. 173–183.

[Lava00] LaValle, S.M., Kuffner, J.J., “Rapidly-exploring random trees: Progress and prospects,” Proceedings Workshop on the Algorithmic Foundations of Robotics, 2000.

[Lato99] Latombe, J.-C. “Motion planning: A journey of robots, molecules, digital actors, and other artifacts,” Int. J. Robot. Res. 18, 11 (1999), 1119-1128.

[Lege00] Leger, C., “Darwin2k, An Evolutionary Approach to Automated Design for Robotics,” 2000, Kluwer Academic Publishers, 0-7923-7929-2

[Loza80] Lozano-Perez, T. “Spatial Planning: A Configuration Space Approach,” IEEE Transactions on Computers, Vol C-32, No. 2, February 1983, pp.108-120. Also, IEEE Tutorial on Robotics, IEEE Computer Society, 1986, pp.26-38. Also, AI Memo 605, December 1980.

[Mess07] Messina, E., “Performance Standards for Urban Search & Rescue Robots: Enabling Deployment of New Tools for Responders,” Defense Standardization Program Office Journal, July/December 2007, pp. 43-48.

[Mich98] O. Michel. “Webots: a Powerful Realistic Mobile Robots Simulator,” Proceeding of the Second International Workshop on RoboCup. LNAI. Springer-Verlag, 1998.

[Nils69] Nilsson, N., “A mobile automaton: An application of artificial intelligence techniques,” Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-69), 1969.

[Pepp07] Pepper, C., Balakirsky, S., and Scrapper, C., “Robot Simulation Physics Validation,” Proceedings of the Performance Metrics for Intelligent Systems (PerMIS) Workshop, 2007.

[Robo09] RoboCup Rescue Virtual League. http://www.robocuprescue.org/wiki/index.php?title=Virtualrobots, accessed 01/15/09.

[Shen05] Xuejun Sheng, “Motion planning for computer animation and virtual reality applications,” Computer Animation, vol. 0, no. 0, pp. 56, Computer Animation 1995, 1995.

[Song01] Song, G., Amato, N. M., “Using motion planning to study protein folding pathways,” Proceedings of the Fifth Annual international Conference on Computational Biology, 2001, pp. 287-296.

[Schl06] Schlenoff, C., Steves, M., Weiss, B., Shneier, M., and Virts, A., “Applying SCORE to Field-based Performance Evaluations of Soldier Worn Sensor Technologies,” Journal of Field Robotics, Vol. 24, No. 8-9, 2006, pp. 671-698.

[Schw83] Schwartz, J. T., Sharir, M., “On the Piano Movers Problem: I. The Case of a Rigid Polygonal Body Moving Amidst Polygonal Barriers”, Communications on pure and applied mathematics, 1983, 36:345-398.

[Schw83a] Schwartz, J. T., Sharir, M., “On the Piano movers Problem II: General techniques for computing topological properties of al-

gebraic manifolds", Advances in Applied Mathematics, vol. 4, 1983, pp. 298-351.

[SWRI] Southwest Research Institute, Applied Physics Division, Micrologger Introduction.

[Syca98] Sycara, K. and Pannu, A. S., "The RETSINA multiagent system (video session): towards integrating planning, execution and information gathering," Proceedings of the Second International Conference on Autonomous Agents, ACM, New York, NY, 1998

[Tayl07] Taylor, B., Balakirsky, S., Messina, E., and Quinn, R., "Design and Validation of a Whegs Robot in USARSim," Proceedings of the Performance Metrics for Intelligent Systems (PerMIS) Workshop, 2007.

[Thom05] Shawna, T., Song, G., Amato, N. M., "Protein Folding by Motion Planning," Physical Biology, 2005, 2:S148-S155.

[Unre08] UNR, Unreal engine, http://www.epicgames.com, accessed June 1, 2008.

[Usar09] USARSim Homepage, http://www.sourceforge.net/projects/usarsim, accessed 01/15/2009.

[VMAC09] VMAC Home page. http://vmac.hood.edu, accessed 01/15/2009.