Development of a Dynamic Mobile Robot Simulator for Astronaut Assistance

P. Heiskanen, S. Heikkilä, A. Halme

Department of Automation and Systems Technology
Helsinki University of Technology
P.O. Box 5500, 02015 TKK
Finland
paavo.heiskanen@iki.fi, seppo.heikkila@tkk.fi, aarne.halme@tkk.fi

ABSTRACT

The need for robotic assistance has been identified to be essential in space exploration missions. This work is targeted for a centauroid robot, called WorkPartner, which is currently being developed to work interactively with astronauts. This paper describes SimPartner, a dynamic robot simulator of the WorkPartner robot by using ODE (Open Dynamics Engine) software. The software incorporates an accurate model of the robot, including part lengths, masses, joints, actuators and sensors. The model can be used in real time to estimate forces and torques that would be difficult to measure from the actual robot, develop robot control code and to predict robot behavior e.g. in tele-operated tasks. The contribution of this paper is to show that it is possible to create a verifiable real-time dynamic mobile robot simulator for a centaur-like mobile service robot. This is achieved by comparing the simulation data with measurements from the actual WorkPartner robot

INTRODUCTION

The necessity for mobile robot simulators has been recognized by several different robotics research groups. Simulation is used in some phase of almost every mobile robot research project [1]. There are several reasons why robot simulators are useful, including:

1) Reduction of development time of the robot control code.
2) Increased quality of the robot control code.
3) Enabling the real-time testing of complex control algorithms which would require additional sensors.
4) Avoiding unnecessary damage to actual robot equipment when testing new control strategies or stability solutions.
5) Simulation of complex systems without having to build them.
6) Studying robot behavior in an unattainable environment.

Previously, several research groups have also built whole simulator packages themselves, leading to robotics specialists concentrating on things that are not essential in constructing a robust robot control code, such as ground contact modelling and impact forces as described in [2].

PREVIOUS WORK

A planetary rover is a mobile robot located in an extraterrestrial environment, exploring its surroundings. Rovers are very practical in planetary exploration and have been used since the Russian Lunokhod 1 landed on the moon in 1970. The extraordinary success of NASAs MER-A and MER-B (more commonly known as Spirit and Opportunity) solidified the role of autonomous rovers in planetary exploration.

According to [3], the downside of the use of autonomously moving rovers is the increased need to test the stability of mechanical solutions, as well as the sensor and actuator hardware and software, and perhaps most importantly the on-board computer system. This increases the need for quality simulation software.

Several mobile robot and planetary rover simulators, including Player/Stage/Gazebo, SimMechanics, ROAMS, RCAST and RCET exist and are being used in robotics research. The majority of simulators developed by researchers are created using open source source rationale to promote platform independence [4], distributed software development to loosen the coupling between different modules [5] and making use of other available open source libraries. However, proprietary simulators such as the ADAMS package [6] and Envision [7] are also used. Based on previous work, it is possible to form a list of features that identifies a good quality mobile robot simulator:

- **On target**
  The software should be developed with the end user in mind, taking into account his needs and wishes.
- **Open Source**
  Open source code is verifiable by members of the scientific community and thus gives more credibility to the tool. Secondly, it is possible to use some of the vast amount of open source libraries available.
- **Modular**
  Modular code makes it possible to use the best libraries available and change them if necessary; it also promotes code reuse.
- **Flexible**
  The software modules should be as flexible as possible, as this encourages other developers to contribute to the code and also improves the general quality of the code.
- **Parametrized**
  Good quality software enables the user to make changes in the way the software operates without the need for recompiling. This can be achieved by placing as many parameters to human readable text files as possible.
- **Platform independent**
  A good quality code should be programmed so that it can be run on different platforms.
- **Real-time ready**
  It should be possible to run the simulator in real time with operator-in-the-loop.
- **Connected to actual hardware**
  It should be possible to connect the simulator to real robot equipment to make hardware-in-the-loop simulations possible.
- **Verifiable**
  The accuracy of the simulator must be stated and user verifiable.

Simulator structure depends highly on the planned usage of the simulator. Normal mobile robot simulators, such as [8] tend to be more general whereas planetary rover simulators can be programmed for very specific tasks, such as determining optimal wheel diameter [9] or axle length [10]. The general structure of a simulator is presented in figure 1.

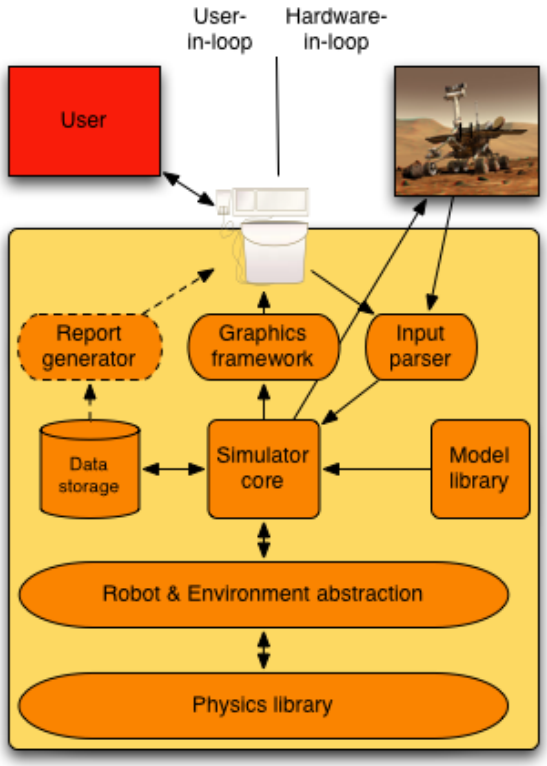

Fig. 1: General mobile robot simulator structure.

Simulators can be used in all phases of a space mission. In the conceptual study and preliminary analysis phase, simulators can be used extensively to test different designs without the inherent cost of building mechanical prototypes. By building a system model and testing it in the simulator, many different design possibilities can easily be tested. Vehicle responses to commands can also be tested by using operator-in-the-loop simulations.

In the design, development and testing phase simulators serve in different roles. When hardware and software choices are made the simulated components can be left out of the model and the actual component response can be tested using hardware-in-the-loop simulations. When the vehicle concept is ready, the on-board software functionality can be tested with just the sensors connected to the simulator. The response of the vehicle to artificial stimuli can thus easily be tested.

In the operations phase the commands that are about to be transmitted to the satellite can be transmitted to a simulator first, confirming the correctness of the commands and verifying that the desired reaction is produced in the vehicle. This reduces the chance of a mission loss due to, for example, a typing error in the control commands.

SIMPARTNER OVERVIEW

SimPartner is an object-oriented dynamic robot simulator which combines several existing open source libraries and technologies to create a versatile simulator framework. The key library used is the Open Dynamics Engine (ODE) that handles the simulation of the rigid body physics in the simulation. SimPartner features a modularized design that allows the user to change parts of the code without having to reprogram the whole simulator. The modules in SimPartner are shown in figure 2.

SimPartner main program, developed in C++, creates an executable application, reading in simulation parameters from a properties file. ODE Model handles communication between the SimPartner main program and the Open Dynamics Engine. SimPartner model of sensors and actuators also include an implementation of the TCP/IP communication stack that allows the user to control the robot and monitor its state through external software. All the necessary data obtained from the physics simulation is stored to a MySQL database if the user so desires.

For efficient control code programming, the user should be able to analyse the accumulated simulation data easily. Fortunately, there are several ways to achieve this due to the wide acceptance of the MySQL database software. For example Matlab has a Database Toolbox that can be used to convert database rows into Matlab workspace variables. This effectively integrates all the analysis power of Matlab to the SimPartner framework.

The simulation environment and the robot are defined using an XML file. The XML file is also validated against a Document Type Definition (DTD) to guide the user to write conforming files that can be interpreted by the XML parser. In addition to a surface plane, SimPartner incorporates a possibility to model terrain features with a heightfield. A heightfield is a data type containing a height value matrix. This matrix stores the height values of the terrain. This matrix is scaled using depth and width values to store information about the size of the field in the simulation environment. Points that reside inside the area but not at the points defined by the field are linearly interpolated by the

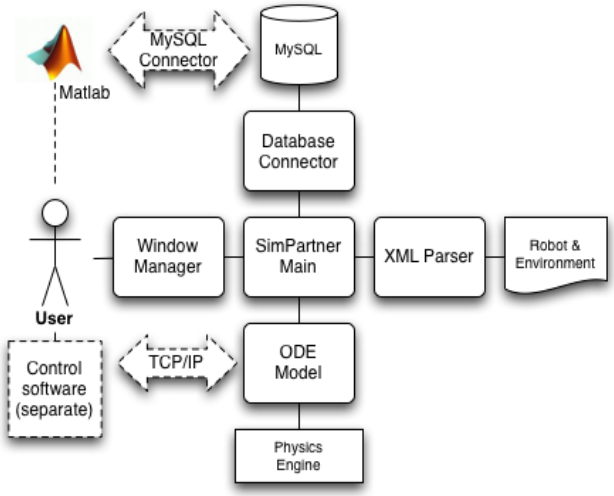

Fig. 2: SimPartner structure

physics engine. Thus, a heightfield is a simple but powerful way for the user to create a terrain with a practically unconstrained accuracy.

A robot contains bodies, joints, sensors and actuators. Bodies are rigid and connected to each other with joints. Sensors and actuators are the only way the user can directly control the robot when the simulation is on-line. Sensors currently implemented include a position sensor and a laser scanner. Actuators are attached to joints and it is the possible to control the joint to move so that a desired position is reached. The joint can also be set to revolve at a constant velocity or to deliver a desired force to the bodies it is connected to, creating an angular motor. Actuators currently implemented include an angular motor and a PID angle controller.

MODELING WITH SIMPARTNER

WorkPartner, a centauroid mobile service robot, was modeled using SimPartner to verify the correct operation of the simulator. The robot was modeled by using the latest CAD model of the robot shown in figure 3. The configuration information and measurements were read from the model and translated into a robot XML file described earlier. Due to the complexity of the model, the robot was modeled in different generations. The idea behind this design approach was to validate the correct functionality of each design iteration before new parts were added.

The first generation model of the robot contained the wheels and supporting structures for the undercarriage. The wheels and other joints have simple motors that can be used to drive the robot. The fact that the wheels of the WorkPartner robot are not steerable causes added complexity to the steering system. In the first generation model, steering was modeled with one hinge joint in the middle of the chassis. The validation test done for this model was a circular trajectory with different inner- and outer wheel velocities to achieve smooth movement. The simulated trajectory was very close to the mathematical model, with the error being approximately 0.5%.

The second generation robot model includes a torso with a simple laser scanner and a position sensor. The torso of the robot was turned with a constant angular velocity. The simulation environment had a wall set at a distance of six meters from the robot center. After an initial jitter, the measurement error grew linearly. This is consistent with the integrator error of the ODE and offers an insight to the best possible accuracy that any SimPartner sensor can achieve. The error levels were also proportional to the distance to be measured, meaning that the longer the distance, the greater will be the relative error.

In third generation, the WorkPartner model included all the essential parts of the actual robot. The validation done to this model was a test in which the robot advanced towards a pole held on the top of two cubes. The robot then picked up the pole, advanced towards another pair of cubes and laid down the pole smoothly so that it remained on top of the other pair of cubes. Force measurements were recorded during the simulation. An interesting detail in the wheel forces can be seen when individual wheel forces are plotted as seen in figure 4. Wheels one and two (in front of the robot) carry a substantially larger amount of the weight of the robot than wheels three and four. Also, the weight of the pole is mostly carried by the front wheels while the load on the rear wheels actually decreases slightly. This can be explained by the

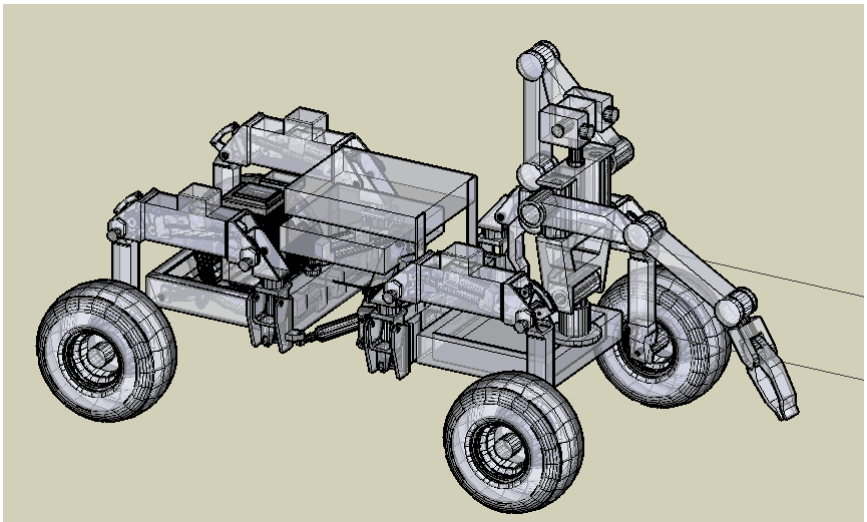

Fig. 3: WorkPartner CAD model

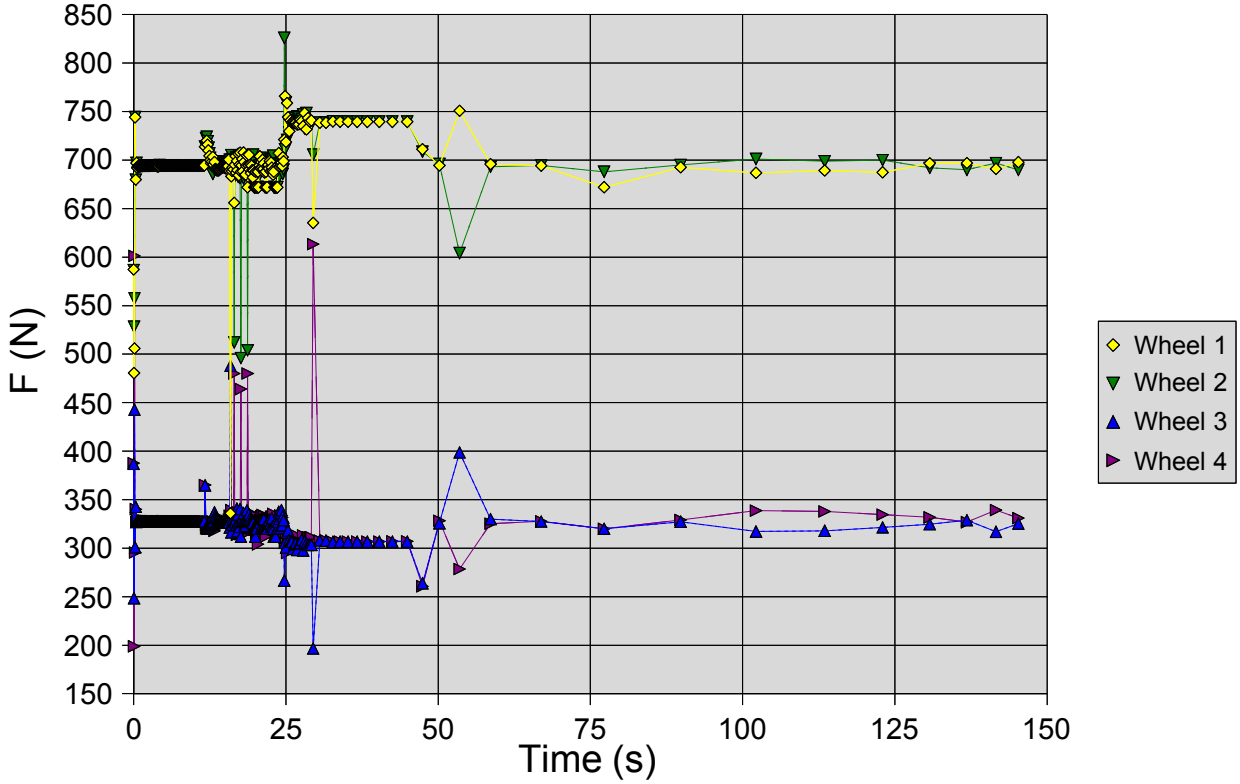

Fig. 4: Forces affecting each wheel in the third generation simulation.

geometry of the robot; the torso and its support structures makes the front end significantly heavier. Actually, this behavior has been observed in the actual WorkPartner robot. The robot is powered by a gasoline engine and batteries which are placed on the rear part of the robot to improve stability by acting as a counterweight for the long manipulators.

The next validation is to verify that the control code can be developed using the SimPartner framework. The scenario is that the robot is placed on a very slippery surface ($\mu = 0.025$) and the goal is to make the robot move as fast as possible. The assumption is that it is possible to use wheel walking to make the robot move with a greater velocity. The initial setup was simply to set the rotational velocity of the wheels to be one radian per second, which will cause the robot to move slowly as the wheels are mainly skidding and only partly moving the robot forward. The next improvement was to use the robot "knee" joints. The simplest way to achieve this was to use caterpillar-like movement where front and rear wheels are moved in turn. The average velocity during the course of the simulation increased dramatically, from 0.0038 to 0.057 meters per second. Next, a rolling walking (rolking) simulation was developed. Rolling walking sequence is based on the idea that the joint motors are used to move the robot with the wheel motors assisting in the movement. Thus only one leg moves at a time. During the movement the wheel of the moving leg is assisting the movement by rotating and the other wheels are kept unlocked. The velocity of the robot was 0.048 meters per second, which is slightly slower than in the previous simulation. Rolling walking with WorkPartner has already been simulated before. [11] describes the wheel forces in a similar locomotion simulation to that presented here. Wheel forces in these two similar simulations proved to be very similar in a strongly dynamic movement.

The final validation was made by comparing SimPartner data to measurements performed using the actual robot. This was achieved by running a simulation similar to that described in [12]. The reference presented the height profile of the terrain used in the test runs made with the actual robot, shown in figure 5. The test runs were made outside during the winter season. This means that the ground was snowy and uneven. The figure mentioned above only covers two dimensions so we can only model the elevation of the ground in one direction. When the WorkPartner robot is traversing the test track, it is using active control for its wheels and legs for energy efficient locomotion. This behavior was not perfectly modeled in the client software as this is not within the scope of this work. After adjusting the mass distribution of the simulation model to match with the actual robot a heightfield was created that corresponds with the terrain used in the test run with the actual robot. The velocity of the robot slows down considerably in the same

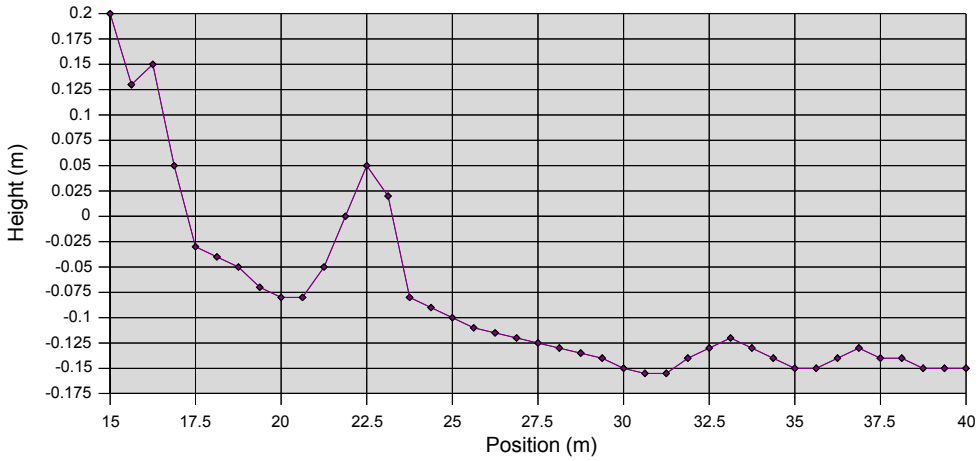

Fig. 5: WorkPartner test terrain

positions as the original test run. It could clearly be seen that these positions correspond to the locations where the terrain causes the robot to slow down. The simulation is thus similar to the actual test drive in this aspect.

The wheel forces from the test run and the simulation are shown in figures 6 and 7. The magnitude of the forces is the same as in the test run with the actual robot, when taking into account the fact that the forces measured from the simulation incorporate approximately 200 newtons of leg and wheel weight. The number of measurements in the simulation is smaller but the spikes in the forces can clearly be seen in the same positions as in the actual test run. The data cannot be compared quantitatively due to the different control algorithms, but it can be said that the simulation resembles the actual test with an accuracy that is enough to validate the simulator.

REMARKS

As stated before, ODE is not at full maturity at this point. There are still several inconsistencies and imperfections in the physics engine that must be taken into account. Fortunately, depending on the type of simulation the user wants to achieve, some of these problems can be remedied by careful tuning of the simulation parameters.

Good example of these inconsistencies that can be observed, especially with box-shaped objects, is object-object penetration. This is basically a failure in the collision detection engine and it is inherent to fixed time-step deterministic physics engines. Object-ground penetration can also be seen and it is based on the same phenomenon as object-object

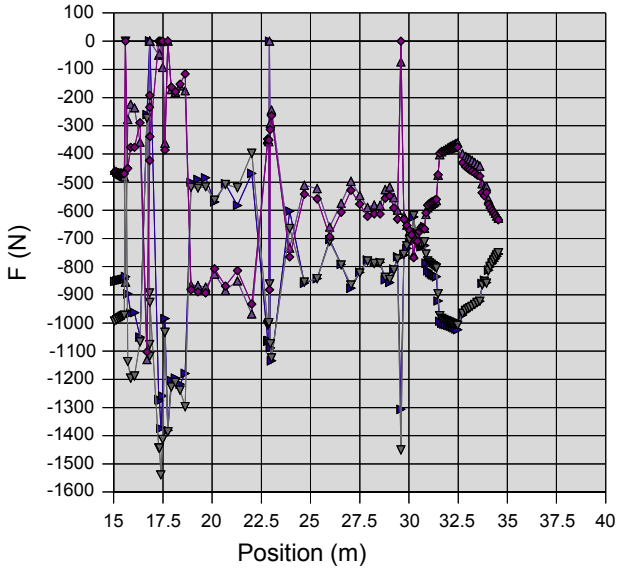

Fig. 6: Wheel forces in the simulation

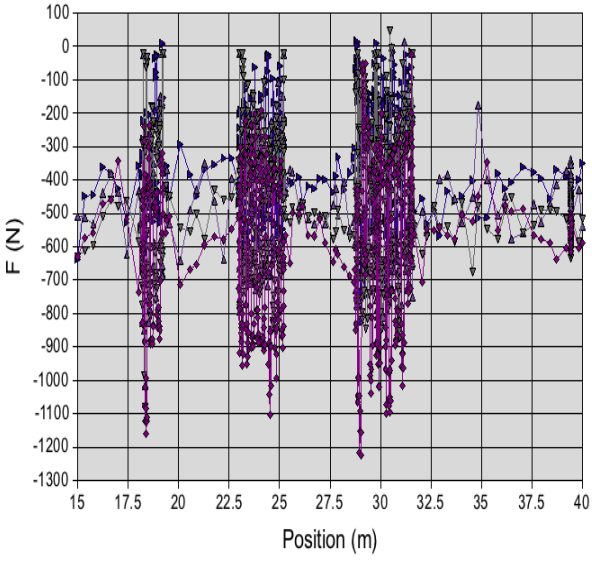

Fig. 7: Wheel forces in the test run

penetration. An implication of this is that the object can come to rest in this position as the supporting surface is calculated from the penetration points. In this case the supporting surface is not formed by the corner points of the object but rather by the intersection points of the ground plane and the penetrated area of the object. In effect, a cube can stand on one of its corners.

Since there are several numerical problems that have to be solved for every simulation step, numerical problems may sometimes occur. One of the most common is the LCP error. LCP stands for linear complementarity problem, a linear algebra problem of finding two vectors that satisfy a certain set of equations based on a square matrix and a column vector. This problem is quite common in optimization, physics simulation and mathematical programming. ODE uses a method developed by [13] .

The multitude of clock calls made by the SimPartner cause the PC clock to drift. This behavior will cause errors of several seconds per one minute of simulation time. This causes severe problems for analyzing the results, although the simulation itself runs well. This problem can be addressed by using NTP to actively keep the clock of the computer in correct time.

The SimPartner framework requires a lot of computing power, especially when database logging is used. If too many other applications are open, and/or the computer does not possess the required computing capabilities, the performance may not be good enough for real-time operation. A decent user experience can be achieved with a 2GHz computer with at least 1GB of memory.

CONCLUSIONS

This paper described the design and validation process of a dynamic mobile robot simulator. The work was based on the state-of-the-art study and literature review of existing simulators and their properties. General software engineering practices and standards of the open source community were also taken into consideration.

The most important validation test was to simulate a full test run done with the actual WorkPartner robot and to compare the results with the data from the actual robot. The test was a success as the results showed that the simulator was able to model successfully the speed and force magnitudes of the real WorkPartner robot. It was also shown that the simulation errors can be identified and mathematically explained to a certain degree.

The SimPartner software framework follows the identified common features of robotic simulators, furthermore the software has been verified to follow mathematical models and validated against actual test data. It can be thus said that the simulator is usable and trustworthy for control code development for mobile robot applications, as long as all the constraints of simulation in general and SimPartner in particular are taken into account. The whole software package is open source and released under the GNU Lesser General Public License. Therefore anyone interested in using or developing the simulator can do so.

It was also shown that real-time performance in dynamic robot simulations is achievable with consumer grade computers. SimPartner was thus validated to be a real-time rigid body physics simulator that can model in real time complex mobile systems in variable terrain.

ACKNOWLEDGEMENTS

This paper is based on the principal authors M.Sc. Thesis done on Helsinki University of Technology supervised by Prof. Aarne Halme and instructed by Mr. Seppo Heikkilä with funding from ESA Directorate of Human Spaceflight, Microgravity and Exploration.

REFERENCES

[1] P. Aarnio, K. Koskinen, and S. Ylönen, “Using simulation during development of combined manipulator and hybrid locomotion platform,” *P*roceedings of International Conference on Field and Service Robotics, pp. 287-294, 2001.

[2] M. Buehler et al., “Stable open loop walking in quadruped robots with stick legs,”. Proceedings. IEEE

International Conference on Robotics and Automation, pp. 2348-2353 vol.3, 1999.

[3] R. Bauer, W. Leung, and T. Barfoot, “Development of a Dynamic Simulation Tool for the Exomars Rover,” i-SAIRAS 2005'-The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space. Edited by B. Battrick. ESA SP-603. European Space Agency, 2005. Published on CDROM., p. 15.1, 2005.

[4] R. Vaughan, B. Gerkey, and A. Howard, “On device abstractions for portable, reusable robot code,”, Proceedings. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2421-2427 vol.3, 2003.

[5] T.H. Collett, B.A. MacDonald, and B.P. Gerkey, “Player 2.0: Toward a Practical Robot Programming Framework,” Proceedings of the Australasian Conference on Robotics and Automation *(ACRA 2005)*, 2005.

[6] J. Fraczek and A. Morecki, “Modelling of contact in walking machines,” IEEE SMC '99 Conference Proceedings, IEEE SMC, pp. 948 - 952, 1999.

[7] P. Aarnio, K. Koskinen, and S. Salmi, “Simulation of the hybtor robot,” Proceedings of the 3rd International Conference on Climbing and Walking Robots, Madrid, Spain: Professional Engineering Publishing Ltd, pp. 267-274, 2000.

[8] N. Koenig and A. Howard, “Design and use paradigms for Gazebo, an open-source multi-robot simulator,”, Proceedings. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2149-2154 vol.3, 2004.

[9] S. Michaud et al., “RCET: Rover Chassis Evaluation Tools,” Proceeding of the 8th ESA Workshop on Advanced Space Technologies for Robotics and Automation, Noordwijk: 2004.

[10] Patel et al., “Rover Mobility Performance Evaluation Tool (RMPET): A Systematic Tool for Rover Chassis Evaluation via Application of Bekker Theory,” Proceeding of the 8th ESA Workshop on Advanced Space Technologies for Robotics and Automation, Noordwijk: 2004.

[11] P. Aarnio, *Simulation of a Hybrid Locomotion Robot Vehicle*, Licensiate Thesis PB2003-101519, Helsinki University of Technology, 2002.

[12] I. Leppänen, *Automatic locomotion mode control of wheel-legged robots*, Dissertation, Helsinki University of Technology, 2007.

[13] R.W. Cottle and G.B. Dantzig, “Complementary pivot theory of mathematical programming,” *Linear Algebra and Its Applications*, vol. 1, pp. 103-125, 1968.