Simulation for Dependable Mining Automation

N Hillier[1] and J Ryde[2]

ABSTRACT

A discussion on simulation tools to aid the development of robotics platforms is presented. The focus of the work is towards mining robotics platforms where operations are typically in unstructured, unpredictable environments and a high degree of dependability in their operations is required. It is argued that a test-based development approach leads to more robust solutions and faster development cycles than traditional single-track field-only development and that the key enabling technology for this development paradigm is the judicious use of simulation tools.

The future direction of simulation for use in online applications such as active planning and as a means to assist tele-operation in high latency communication applications is also briefly considered together with the requirements for a useful simulation environment in terms of the simulator architecture.

Finally, a case study is presented on the application of the discussed methodologies to an unmanned ground vehicle (UGV) for earth-moving tasks, which is being developed in an accelerated time frame with a transient development team.

INTRODUCTION

Mining automation is a difficult domain for robotics development, with significant challenges in development and operation due to the unstructured environments (both geometrically and in material properties), common use of multiple sensing modes and the high degrees of freedom inherent in most experimental mining robotics[3] platforms. Additionally, there are pragmatic difficulties associated with the development of mining robotics platforms stemming from long set-up and shut-down times (pre- and post-start checks) and access to suitable test sites that do not occur with most indoor and small platform robotics.

These issues are further compounded by those applications wherein dependability of the robot is essential due to any number of commercial, remote deployment and/or risk constraints.

This paper argues that a successful development paradigm for mining robotics includes significant use of simulation environments and that certain aspects of the simulation engine architecture allow for more rapid and flexible development of the simulation itself. For clarity, the discussion here is typically constrained to the simulation of robotics in a full three-dimensional (3D) environment.

Furthermore, the authors consider that robotics is a singular field in the engineering disciplines in that testing on the developed product is currently by far the predominant means of development. Most (if not all) other engineering fields involve significantly more extensive use of simulation or numerical modelling tools before prototype and production testing. There are many reasons for the slow uptake of such tools in robotics, not the least being that the popular approach of replaying log files is sufficient for the development and testing of many robotics algorithms and that the appropriate simulation tools are only in their infancy.

The work in Weatherly *et al* (2006) and Grabowski *et al* (2006) details the software architecture and core development processes used in the development of a 2005 DARPA Grand Challenge entrant (the MITRE Meteor). The 2005 Grand Challenge was a DARPA (United States of America's Defence Advanced Research Projects Agency) sponsored event requiring autonomous off-road vehicles to

1. Research Scientist, CSIRO, PO Box 883, Kenmore Qld 4069. Email: nick.hillier@csiro.au

2. Post-Doctoral Researcher, CSIRO, PO Box 883, Kenmore Qld 4069. Email: Julian.ryde@csiro.au

3. In this work, the authors make a distinction between mining automation and mining robotics, in that robotics includes some element of mobile actuation (such as a robotic arm, or an automated vehicle), whilst automation covers a wider field, including robotics but also process automation and the like.

travel a 132-mile desert route in under ten hours without any external assistance. These works highlight a number of advantages and also a significant caveat to the use of simulation heavy field robotics development and so a summary is presented here to introduce some key concepts.

The fundamental restrictions based upon this team were:

- accelerated development (concept to deployment was approximately 11 months),
- no previous Grand Challenge experience,
- no immediately local off-road test site (the authors claim that in ten months, only ten days of testing was conducted outside of a parking lot), and
- a relatively small development team (five core members).

The team chose to use a simulation heavy development cycle including full ten hour race simulations to test software robustness as well as data-replay from limited off-road real-world testing. Both the simulated and replayed data interfaces were transparent to the code operating the real robot, a concept that can readily be achieved through the use of modern robotics middleware (eg the Robotic Operating System (ROS) (Quigley *et al*, 2009), Dynamic Data Exchange (DDX) (Corke *et al*, 2004) and Player/Stage (Gerky, Vaughan and Howard, 2003)). The software was also transparent to real-time and simulation-time concepts to overcome flexible computation availability during simulation for faster and slower than real-time execution without timing problems.

Ultimately, the Mitre Meteor DARPA Grand Challenge entry passed all the qualification rounds to start the final robot race (only 23 of the original 195 entrants started the desert race); however, it ultimately failed due to misinterpretation of dust as an obstacle.

This highlights a fundamental issue for simulation driven development: the question of the appropriate level of simulation fidelity and the simulation versus field testing trade-off. The MITRE team described the increase in model fidelity due to the incorporation of field testing data as a 'lasting repository of project experience', and the take away message is that simulation development and testing alone is insufficient for practical applications; however, its judicious use can significantly speed up the development cycle for field-based robotics.

BRIEF SUMMARY OF THE STATE-OF-THE-ART IN ROBOTICS SIMULATION

Simulators

The robotics community has recently seen rapid development of a number of capable 3D, physics enabled simulation platforms. Beyond the ever-growing list of custom simulation solutions that institutions are developing for their own one-off systems (for example the abundance of simulators for the RoboCup four legged league), those that the authors' consider to be the most useful or seem to be currently the most popular for robotics research are very briefly described in Table 1.

Amongst the numerous offerings available, the OpenRAVE platform is currently the tool of choice for the authors. Features that are critical to the authors' work, provided by this solution, which stand out from the others include:

- Simple, flexible interfaces in a variety of languages (C++, Octave/Matlab and Python).
- A dynamically loadable plug-in type architecture that allows for significant customisation of very low-level interfaces (such as the employed physics engine and collision checking) and ease of integration into middleware (comes with interfaces to ROS, and the authors trivially added support to DDX as required).
- A modular design providing an opportunity for a light-weight customised version to be run live on platforms with less processing power via replacement of almost any component.
- Platform independence (support for Windows, MacOS and Linux) with the transparency of open source code.
- The ability to dynamically create, destroy and change the geometric properties of items in the simulation environment. This is a particularly powerful option for operations that manipulate the environment.

Furthermore, the OpenRAVE environment allows for environment cloning, a powerful feature that could be exploited for more advanced simulation tasks such as parallel execution for run-time planning tasks, although the authors are yet to exploit this feature.

TABLE 1
Summary of key (comparable) simulator features of some more popular robotics simulation packages.

Simulator	Employed Physics	Source	Platform
Gazebo (Koenig and Howard, 2004)	ODE	Open	Linux
MS VSE (http://msdn.microsoft.com/enus/robotics/)	PhysX	Closed	Windows
Unity (http://unity3d.com)	PhysX	Closed	Windows and MacOS
USARSim (Carpin *et al*, 2007; Zaratti, Fratarcangeli and Locchi, 2007)	Unreal Engine	Sim open, Engine closed	Windows, MacOS and Linux
OpenRAVE (Diankov and Kuffner, 2008)	Any (ODE by default)	Open	Windows, MacOS and Linux
OpenHRP (Kanehiro, Hirukawa and Kajita, 2004)	Various custom algorithms	Open	Windows and Linux
Webots (Michel, 2004)	ODE (customisable in 'Pro' version)	Closed	Windows, MacOS and Linux
OpenSimulator (http://www.opensimulator.org)	ODE (+ others)	Open	Windows, MacOS and Linux
Matlab and similar tools (http://mathworks.com)	Custom algorithms	Closed	Windows, MacOS and Linux

Applications in automation

The focus of this paper is on the use of simulators for algorithm and system development, by emulating a robotic platform and its environment in a virtual world. Although this is probably the most well known application of simulation tools in robotics, there are some other notable applications that may not be as widely known.

The competitive robotics industry has adopted simulation use wholeheartedly, with a wide range of robotics competitions sporting either a simulation component, alternative stream (eg RoboCup's simulation leagues) or having no real-world component at all (eg Rat's Life is a pure simulation competition, although the equivalent hardware can be purchased). This is extremely beneficial to the robotics community as simulation competitions significantly reduce the financial barriers to entry into the industry by removing the associated costs and skills required to purchase and maintain robotic hardware and supporting infrastructure.

The advent and rise of augmented reality interfaces (Milgram and Kishino, 1994) is boosting work in the area that some refer to as augmented virtuality, a field which is particularly prevalent in tele-operation applications with large latencies. In these applications, consistency of the user experience is provided through a virtual interface consisting of a GUI and a simulation back-end that synchronises with the real-world when possible. The simulation engine is then used to extrapolate the system state and present the user with a real-time interface.

The use of simulation in robotics development is not restricted to software-only manifestations. Hardware-in-the-loop (HIL) simulation has been used extensively in many industries as a developmental and debugging tool. In robotics, it is often used as part of a staged development cycle, whereby sensor performance, timing issues or control demands require critical testing before deployment. HIL simulation often acts as a substitute for sensory information, particularly where an appropriate real-world test environment is difficult to achieve (eg off-world Rupp *et al*, 2009) or is deemed high-risk in the case of potential failure (eg aerial systems).

Simulators are now also being increasingly used in a hybrid manner to artificially increase robotic populations. A typical example of this is in the field of self-reconfigurable (or modular) robotics, whereby one may wish to evaluate system performance of many tens, hundreds or thousands of components, but have hardware for only a few tens of modules (see for example, the HIL simulation work of Lal and Fitch, 2009).

There is also scope for simulation tools to be used within an automated machine's programming paradigm in an online manner, particularly with respect to planning tasks. This is often referred to as decision support. Here, the robot holds some model of the world and is able to run a simulation of the robot's execution of the plan in a faster than real-time manner to evaluate various decision metrics.

The application of the virtual world provided by simulation to robotics is almost endless. Some of the other uses of which the authors are aware include visualisation, training (of both operators and for evolutionary type systems) and as a tool for the design of user interfaces.

Each of the above presented applications for simulation have different performance criteria and ideally a simulation engine should be sufficiently flexible to allow reuse in all of the above scenarios with minimal customisation.

TEST-BASED DEVELOPMENT

System dependability (reliability, trustworthiness) is a key criteria for the development of robotics within many groups due to any number of commercial, remote deployment and/or risk constraints.

In practice, there appear to be three streams of development methodology in use:

- restriction of development tools, algorithms and platforms to those that are time-proven and tested;
- verification via theorem proof (see the methodologies of Bensalem, Ingrand and Sifakis, 2008); and/or
- thorough, test-based development.

Given the current pace of worldwide robotics research and the rapid development of new algorithms that can be considered crucial to the performance of advanced mining automation systems (eg simultaneous localisation and mapping – SLAM, computer vision), the authors consider that the achievement of the dependability criteria is best developed through test-based methodologies.

Such methodologies also present a measurable metric upon which we can assess the system's dependability. Furthermore, test-based development allows for aggressive re-factoring and experimentation without fear of a regression in functionality. Finally, the test-based development methodology has a demonstrated track record in other disciplines.

Practicalities for automating machinery

Test-based development has been a very successful paradigm in software engineering with various derivatives such as test driven development seeing success (see for example Nagappan *et al,* 2008). It has the potential to be beneficial for robotics development; however, the translation of this paradigm to the robotics domain can be problematic.

While it may be argued that test-based development is possible with indoor robots, it is often impractical for application to the mining domain for a number of reasons:

- Tests on real robots have to run in real time by definition, this is time consuming for the development cycle of test-based development, as such cycles consist of multiple iterations of writing test code, writing program (deployment) code and running tests.
- The repeatability of the test suite is important to the methodology's success. Ensuring that the test harness and robotic system are in the same initial state before testing can be difficult in real mining robotics deployments, but is readily achievable for a software simulator.
- While regression testing in software engineering has negligible cost, the regular execution of regression tests on autonomous systems can be costly in terms of fuel consumption and mechanical wear.

Performing such tests for automated machines can be segmented into two distinct categories:

1. Those tests that exercise passive algorithms, and can be run on recorded data, such as simultaneous localisation and mapping (SLAM), where the system is an observer of the data.
2. Those tests that exercise active algorithms, whereby the sensory data is altered by the action of the algorithm (the system is a participant in the creation of the data – eg path planning and control execution). Such tests are required to be run on the robot (live) or through some element of simulation.

We argue that simulation is typically the only practical means by which to exercise active tests, and typically also allows for the exercising and evaluation of passive algorithms. Additionally, a simulator allows for fully automated testing at any time, which is vital for regression testing and rapid development.

Ideally, tests should be deterministic and idempotent, namely, that $f(x) = f(f(x))$. It is important to appreciate that success in simulation is most often a necessary but not sufficient condition for success in the field. For this reason, simulation tests should be run following all code changes and before every operation in the field.

Furthermore, there is an array of tests that are substantially easier to write in simulation due to the ready availability of expected results in the simulator internals (eg those that require or generate pose estimates). Some may argue that such tests are not productive, because they do not easily translate to implementation on deployed robots (the truth is not readily available); however, the mere existence of the test and the ease with which it can be used for underlying algorithm development purposes can greatly accelerate the development time frame and again, provides some increase in the measure of system dependability.

SIMULATION FIDELITY

The final aim of the use of the simulator in robotics development is to maximise the intersection of code that can be run in the real-world and simulator. If such a system can pass all required tests in the simulation environment, this infers that a high level of trust can be placed in the deployed code. The issue of simulation fidelity is then raised. We refer to simulation fidelity as the level of detail and accuracy to which the simulation reflects the real system; for example, what sensor models are used or the accuracy of the physics (or indeed, if a kinematic only simulation is sufficient).

As touched upon in the introductory discussion with reference to the MITRE Meteor DARPA Grand Challenge entrant, the fidelity with which the simulation reflects the real world system(s) can both be crucial to the success of the robot and a valuable log of the development process.

Whilst it is easy to argue that one should aim for a simulation environment with the lowest possible complexity (and hence minimal fidelity) required for the task at hand (this reduces the possible number of failure points in the simulation, easing debugging and ensures that time is not spent developing redundant functionality in the simulation environment). The ready availability of physics simulation engines gives a base level of fidelity that can aid in identifying unforeseen, but common execution failures. An example of this is the classic planar world assumptions, which the authors found to be immediately impractical on the UGV discussed earlier, due to the pitch and roll effects of the platform in acceleration manoeuvres and the associated errors in scanning laser returns when mapped to a two-dimensional (2D) environment.

The issue of required fidelity for the task at hand has implications on the choice of simulation package. Where by default the simulator is lacking the minimal required fidelity, the simulation engine must be sufficiently flexible to allow the researchers to increase the fidelity of the simulation environment as required.

TIMING

Beyond the fidelity issues of non-simulated phenomena that may be critical to the robots success (or failure), the authors argue that the other most critical element to ensuring the successful transfer of simulator developed code to real-world robotics is that of timing. Simulators offer the ability to control the passage of time; however, this introduces a wide array of possible failure mechanisms to the system that may not be apparent in systems developed without the use of a simulation environment.

Controlling time

The control of the passage of time is a powerful tool only available in simulation that, amongst other advantages, enables real-time debugging. Conventional debugging pauses the execution to allow inspection and possible modification of the program state. For systems operating in the real world such debugging practice is typically impossible, as whilst the program execution is paused, time in the real world continues (pausing the system in itself typically alters the system state, and in some instances may induce catastrophic failure – eg debugging a helicopter control system during flight). Such debugging tools, however, are easily applied in simulation by pausing both the simulator and the robot program when a break condition is reached. Altering simulator time can also aid with visualising actions that would otherwise occur too fast in reality. Without simulators these sorts of debug actions are typically done by real-time configurable logging and post mission inspection of log files.

Accelerating simulation time also allows the execution of experiments that would otherwise be difficult to run. An example is that of training a neural network or an evolutionary system. Once the algorithms have been trained in simulation they require less verification and intervention before deployment on the real robot.

Simulation time-induced failures

The alteration of time in the simulation environment, whether intentional as outlined previously, as a side-effect of under-specified hardware or a by-product of the discrete time-stepping employed can reveal a number of failure modes. When properly addressed, a significantly more robust robotic solution is developed.

These variations in time affect the independence of processes and computation nodes and reveal any dependence of algorithms on clock timing. Care must be taken that the time reference provided by the simulation and that used in the operational code is consistent. Furthermore, if computing resources are scarce on the deployed platform, there arise issues surrounding slower and faster than real-time simulation, whereby required computations may use all available resources and unpredictable behaviour may result.

CASE STUDY – SKID STEER LOADER UNMANNED GROUND VEHICLE

This section discusses a few uses of simulation tools in the development of an autonomous skid steer loader (modified Bobcat S185) displayed in Figure 1. This is a versatile and ubiquitous vehicle typically deployed on construction sites. Its main defining characteristics are its compact size, ability to turn in place and the positioning of the operator between the boom arms. The platform has significant operational safety risks induced by high turn rates, large actuation forces and high moving mass.

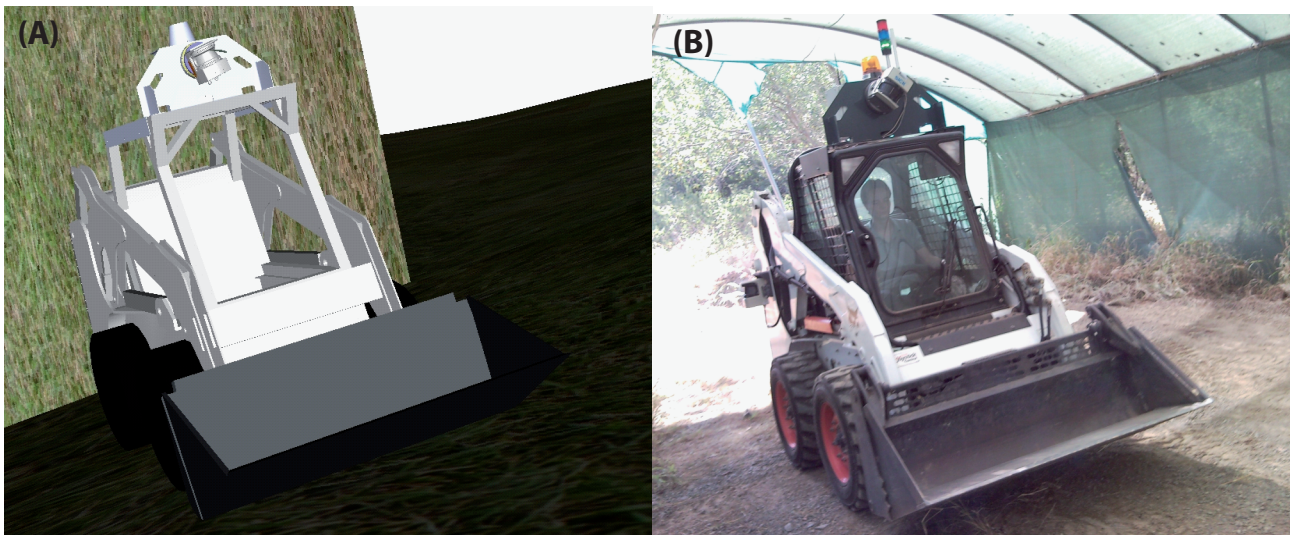

FIG 1 - (A) Implementation of the bobcat simulator in OpenRave; (B) the real bobcat.

Sensor noise failure

During early platform development, a control system failure occurred and was found to be due to harmonic noise on a velocity sensor. This manifested in the UGV drifting in position when commanded to hold a static position. An increase in simulator fidelity was made by implementing a capped random walk as noise to the velocity sensor. The addition of this type of noise to the simulated sensor resulted in the simulated UGV exhibiting similar behaviour to the real platform. A simple station hold test was written and the controller adjusted to account for the sensor noise. The existence of this sensor noise and associated test has resulted in a system that allows for the identification of future regression in this control functionality and allows for increased confidence during control system code re-factoring.

We have since extended this type of testing to include adjusting and increasing sensor noise levels to identify when algorithms fail, and what level of sensing accuracy is required.

Sensor timing analysis

As part of the simulator development, a number of issues related to timing became apparent (similar to those discussed in section timing). Figure 2 shows a comparison of the differences between various sensor timestamps as reported by the system middleware. There is a significant variation with some interesting multi-modal distributions in the real UGVs reported timestamps. Those reported by the simulator are much tighter in tolerance and closer to what we would expect (see the bi-modal and tri-modal distributions of the scanning laser sensors (lmsLeft, lmsRight, Spin0) and the GPS sensor (novatelGPSPos, novatelGPSVel)). We know that such sensors report very regular information, yet this is lost between the sensor and the provision of a timestamp in our middleware. The source of the error is unclear as other sensor data (eg the inertial solution (novatelINS) and control demands (lllDemand)) is reported within a tight distribution. These timing issues may become critical in the future and efforts are underway to understand such errors.

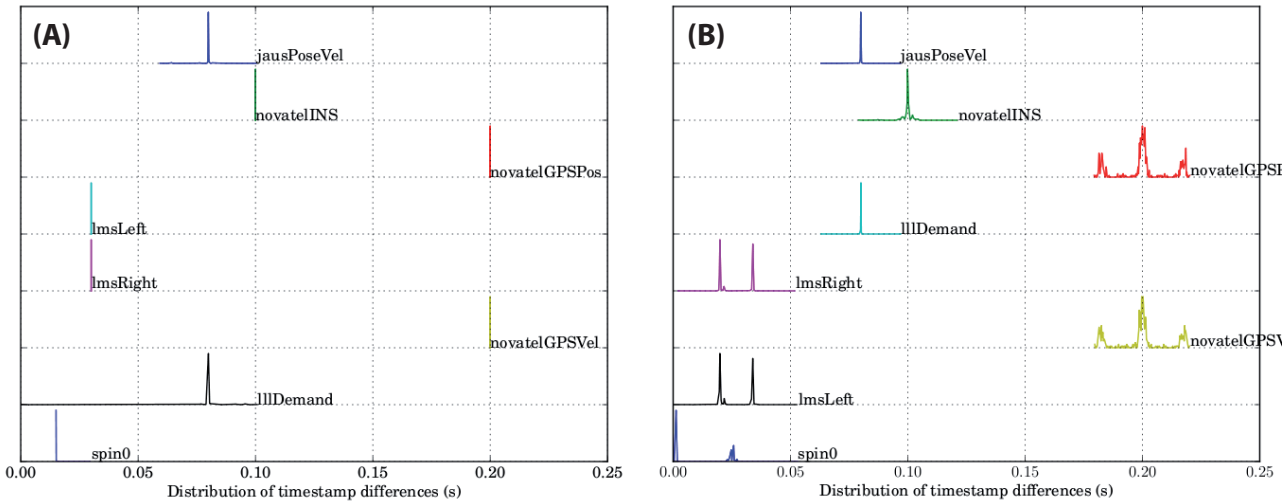

FIG 2 - (A) Comparison of time deltas between updates for simulation; (B) the real unmanned ground vehicle.

Trajectory comparison

As part of the development of way-point following code, a reduced set of the UMBmark odometry benchmark test (Borenstein and Feng, 1995) was performed via way-point execution (a bi-directional square path). The results of this test are presented in Figure 3. The simulation environment was used to develop the way-point code almost exclusively with only minor alterations following field

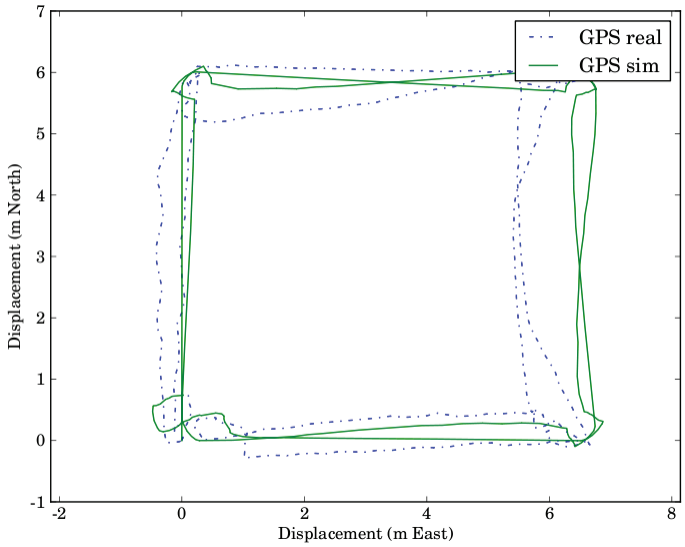

FIG 3 - Comparison of the RTK-GPS antenna trajectories measured experimentally and in simulation for a simple bi-directional square path, starting at the origin and moving north first.

testing. It can be seen that the four corner way-points are consistently reached both in simulation and in the experimental data set. However the simulation lacks an appropriate level of fidelity in the modelled terrain – there are significant altitude changes (0.5 m over the 6 m sides of the square) and much higher roughness in the test area as opposed to the perfectly flat and uniform surface of the simulation. These non-uniformities in the experimental test site's surface perturb the controller and result in the poor tracking performance. Efforts are under way to import point-cloud terrain information into the simulation space.

Nonetheless, the simulation provides easy access to the true trajectory in all circumstances. Typically, for field deployment, RTK GPS information is the closest (position error around 0.05 m) to the true trajectory available and in some test sites, even this measure is unreliable due to multi-path effects. This testing has also prompted the development of a trajectory following controller rather than the simple way-point code currently employed, wherein access to the true trajectory is highly beneficial to the controller's development.

Earth moving demonstration

The first significant project milestone for this UGV is the demonstration of a material moving task. This task involves autonomously moving material from one location to another and is executed in both reality and the simulation environment. The simulation of earthen material is typically not an implemented feature of robotics simulators, and whilst work on accurately modelling earthen material in simulation for robotics has been conducted (Halbach, 2007; Cleary, 1998), we chose to use a number of 0.1 m sided cubes as an approximate representation.

The authors believe that the use of the simulator significantly reduced algorithm test and development time as well as the required skill level to get a high-risk robotic platform conducting useful tasks. We estimate development time to take a platform with open-loop, normalised control inputs to demonstrable performance of an earth-moving task at under 900 hours. Development time spent customising the simulation environment for this task (as opposed to learning and developing the simulator itself) is estimated at a further 100 hours.

CONCLUSION

It has been argued that test-based development for mining robotics provides benefits such as:

- tests allow for aggressive refactoring and experimental algorithms development without fear of regression in functionality, and
- a higher confidence in software system dependability is achievable.

Simulation is an enabling technology for test-based development. The key issues in using simulation for algorithm development and testing for mining robotics applications are:

- *Fidelity*: does the simulation provide sufficient fidelity to be a useful tool (especially for testing)? Is the architecture of the simulation engine appropriately designed for the developers to alter the simulation fidelity as required, not only to change the representation of the physical world, but also to implement a common low-level interface for code and maximise the intersection of code that is run on the simulator and the real robot. This flexibility allows the users to better examine edge-case scenarios and the robustness of developed code.
- *Timing*: does the code framework, including the simulator, middleware and extensible code elements have sufficient capability to handle non-real-time execution (if desired, or otherwise is there sufficient resourcing available to guarantee real-time execution)? Running faster or slower than real-time and the ability to pause both program execution and the simulation environment concurrently provides a powerful debugging tool and a means to artificially increase a learning or evolutionary robot's exposure to experience.

We have presented some results from a basic exemplar of the test-based development methodology's use in the mining robotics domain. Further to the results presented, the authors believe that the approach leads to accelerated progress and that the test structure gives a more robust development process.

ACKNOWLEDGEMENTS

The authors would like to thank Leon Stepan and Hendrik Erckens, who provided much of the control system development and field trials for the UGV work presented; the Engineering Support

team from the CSIRO Autonomous Systems Lab, who provided the automation interfaces to the bobcat platform and the CSIRO's Minerals Down Under Flagship, who provided funding for the work presented.

REFERENCES

Bensalem, S, Ingrand, F and Sifakis, J, 2008. Autonomous robot software design challenge, in *Proceedings Sixth Joint Workshop on Technical Challenge for Dependable Robots in Human Environments* (IARP/IEEE-RAS/EURON: Pasadena).

Borenstein, J and Feng, L, 1995. UMBmark: A benchmark test for measuring odometry errors in mobile robots, in *Proceedings SPIE Conference on Mobile Robots*, pp 113-124 (SPIE: Philadelphia).

Carpin, S, Lewis, M, Wang, J, Balakirsky, S and Scrapper, C, 2007. USARSim: A robot simulator for research and education, in *Proceedings International Conference on Robotics and Automation*, pp 1400-1405 (IEEE: Rome).

Cleary, P W, 1998. The filling of dragline buckets, *Mathematical Engineering in Industry*, 7:1-24.

Corke, P, Sikka, P, Roberts, J and Duff, E, 2004. DDX: A distributed software architecture for robotic systems, in *Proceedings Australasian Conference on Robotics and Automation* (ARAA: Canberra).

Diankov, R and Kuffner, J, 2008. OpenRAVE: A planning architecture for autonomous robotics, Robotics Institute, technical report, CMU-RI-TR-08-34 [online]. Available from: <http://openrave.programmingvision.com> [Accessed: 10 September 2010].

Gerky, B P, Vaughan, R T and Howard, A, 2003. The player/stage project: Tools for multi-robot and distributed sensor systems, in *Proceedings International Conference on Advanced Robotics*, pp 317-323, Coimbra, Portugal.

Grabowski, R J, Weatherly, R M, Bolling, R H, Seidel, D, Shadid, M and Jones, A, 2006. MITRE meteor: An off-road autonomous vehicle for DARPAs grand challenge, *Journal of Field Robotics*, 23(9):811-835.

Halbach, E, 2007. Development of a simulator for modeling robotic earth-moving tasks, Master's thesis, Luleå University of Technology, Scandinavia.

Kanehiro, F, Hirukawa, H and Kajita, S, 2004. OpenHRP: Open architecture humanoid robotics platform, *International Journal of Robotics Research*, 23(2):155-165.

Koenig, N and Howard, A, 2004. Design and use paradigms for gazebo: An open-source multi-robot simulator, in *Proceedings International Conference on Intelligent Robots and Systems*, pp 2149-2154 (IEEE/RSJ: Sendai).

Lal, R and Fitch, R, 2009. A hardware-in-the-loop simulator for distributed robotics, in *Proceedings Australasian Conference on Robotics and Automation* (ARAA: Sydney).

Michel, O, 2004. Cyberbotics Ltd – Webots™: Professional mobile robot simulation, *International Journal of Advanced Robotic Systems*, 1(1):40-43.

Milgram, P and Kishino, A F, 1994. Taxonomy of mixed reality visual displays, *IEICE Transactions on Information and Systems*, E77-D(12):1321-1329.

Nagappan, N, Maximilien, E M, Bhat, T and Williams, L, 2008. Realizing quality improvement through test driven development: Results and experiences of four industrial teams, *Empirical Software Engineering*, 13(3):289-302.

Quigley, M, Conley, K, Gerkey, B, Faust, J, Foote, T B, Leibs, J, Wheeler, R and Ng, A Y, 2009. ROS: An open-source robot operating system, in *Proceedings International Conference on Robotics and Automation, Open-Source Software workshop (IEEE)*.

Rupp, T, Boge, T, Kiehling, R and Sellmaier, F, 2009. Flight dynamics challenges of the German on-orbit servicing mission DEOS, in *Proceedings 21st International Symposium on Space Flight Dynamics*, Toulouse, France.

Weatherly, R M, Kuhl, F S, Bolling, R H and Grabowski, R J, 2006. The Mitre Meteor robot control software: Simulate as you operate, in *Proceedings Winter Simulation Conference 2006*, pp 1294-1298, Monterey, California.

Zaratti, M, Fratarcangeli, M and Locchi, L, 2007. A 3D simulator of multiple legged robots based on USARSim, in *Proceedings Tenth International Robocup 2006 Symposium*, pp 13-24 (Springer).