Quantitative assessments of USARSim accuracy

S. Carpin, T. Stoyanov, Y. Nevatia
School of Engineering and Science
International University Bremen
Bremen, Germany

M. Lewis, J. Wang
Dept. of Information Sciences and
Telecommunications
University of Pittsburgh
Pittsburgh, USA

Abstract— **Effective robotic simulation depends on accurate modeling of physics and the environment as well as the robot, itself. This paper describes validation studies examining feature extraction, WaveLan radio performance, and human interaction for the USARSim robotic simulation. All four feature extraction algorithms showed strong correspondences between data collected in simulation and from real robots. In each case data extracted from a well lit scene produced a closer match to data extracted from a simulated image than to camera data from a poorly lit scene. The radio simulation also performed well in validation showing levels of attenuation due to intervening walls that were similar to signal strengths measured in the modeled environment. The human-robot interaction experiments showed close correspondence between simulator and robots in performance affected by robot model, control mode and task difficulty.**

I. Introduction

USARSim is a high fidelity robot simulator built on top of a commercial game engine [1] with a wide range of possible applications. USARSim is currently being used to investigate human robot interfaces (HRI), to develop and tune robot algorithms, and to study cooperative behaviors. USARSim has recently been adopted by the Robocup Federation [2][3] as the software infrastructure for a new Urban Search and Rescue (USAR) competition that models robots and environments from the USAR Robot League. It joins an earlier Robocup Rescue simulation that focuses on a higher level of logistics and emergency management. Although robot simulators have been widely used since the field's inception there remain widespread reservations about their usefulness. There are a variety of reasons behind these concerns. First, robot simulators have often offered application program interfaces that were inconsistent with those found on real robots. This made it difficult to move software between robot and simulator for code development and debugging which was often the primary purpose for using simulation. This problem has been largely overcome by hardware neutral middleware such as the widely used player/stage software [4][5]. A more damaging criticism concerns discrepancies that may be found between results obtained from simulation and those obtained with real robots. A prime tenet of modern behavior-based robotics [6] is that effective systems can be designed by eliminating internal representations and focusing instead on the direct relation between stimulus and action [7]. From this perspective a good simulation must simultaneously supply an accurate model of the robot's geometry and kinematics, accurate models of sensors, an accurate model of the environment, and an accurate model of the robot's interaction with that environment. If any one of these constituents breaks down the simulation can no longer provide an adequate model of the process being studied. Simulation requirements were far more relaxed for an earlier generation of robots that relied on planning and many robot simulators still provide only schematic or 2D models of the environment and pay little attention to the physics of the interaction between robot and environment. USARSim, by contrast, provides detailed models of both the environment and the physics of interaction making accurate simulation for behavior-based robotics a possibility.

In this paper we provide a quantitative evaluation of the accuracy of USARSim, paying particular attention to the validation of robot performance, as well as the perceptual processes. Specifically, we define a set of perceptual tasks to be studied both in simulation and in reality, as well as metrics to compare the obtained results. The goal is to provide quantitative indices that indicate to which degree it is possible to extrapolate results obtained in simulation. Additional validation data are reported for disruption of radio communications and human control of robots. The overall USARSim architecture is described in section II, with an emphasis on the specific components devoted to perception and action. One of the tasks more relevant in mobile robotics is visual perception. Section III presents a set of algorithms commonly used for robotics oriented image processing, as well as performance indices. In multi-robot systems, inter-robot communications based on wireless channels play a relevant role, but up to now few simulators explicitly model aspects like signal degradation and the like. These topics are addressed in section IV. Section V presents data for two robots controlled by operators using two control modes showing correspondences in behavior between simulated and real robots. Finally, conclusions are offered in section VI.

II. USARSim software architecture

USARSim uses Epic Games' UnrealEngine2 to provide a high fidelity simulator at low cost. The current release consists of models of standardized disaster environments, models of commercial and experimental robots, and sensor models. USARSim also provides users with the capability of building their own environments and robots. Its socket-based control API was designed to allow users to test their own control algorithms and user interfaces without additional pro-

gramming. USARSim includes detailed models of the NIST Reference Test Arenas for Autonomous Mobile Robots [8] and offers the possibility of providing more realistic challenges and significantly larger disaster environments.

The official release of USARSim available from (www.sourceforge.net/projects/usarsim) currently provides detailed models of eight robots including both experimental and commercial robots widely used in USAR competition. These models were constructed using the Karma physics engine [9], a rigid body simulation that computes physical interactions in realtime. A hierarchy of sensor classes have been defined to simulate sensor data. Sensors are defined by a set of attributes stored in a configuration file, for example, perception sensors are commonly specified by range, resolution, and field-of-view.

The scenes viewed from the simulated camera are acquired by attaching a spectator, a special kind of disembodied player, to the robot. USARSim provides two ways to simulate camera feedback: direct display and image server. Direct display uses the Unreal Client, itself, for video feedback, either as a separate sensor panel or embedded into the user interface. While this approach is the simplest, the Unreal Client provides a higher frame rate than is likely to be achieved in a real robotic system and is not accessible to the image processing routines often used in robotics. The image server intermittently captures scenes in raw or jpeg format from the Unreal Client and sends them over the network to the user interface. Using the image server, researchers can tune the properties of the camera, specifying the desired frame rate, image format, noise, and/or post processing needed to match the camera being simulated.

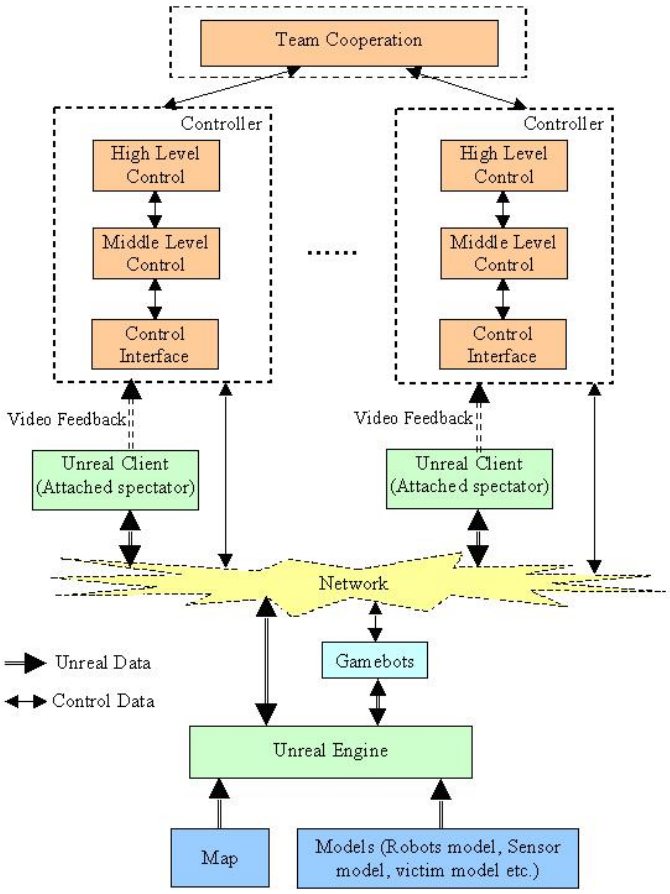

Fig. 1. System Architecture

III. Validation of vision in USARSim

Vision is one of the richest perceptual sources for both autonomous and remotely operated robots. A realistic simulator cannot therefore omit a realistic and quantitatively precise video simulation component. Within USARSim, video input is produced by directly grabbing images from the scene rendered by the visualization component of the game engine. Frames are provided to the robotic controller encoded as jpegs of different quality and with different resolutions. We have implemented four different image processing algorithms that require the fine tuning of several parameters. The parameter fine tuning phase has been performed exclusively in simulation and then the same algorithms have been run on real world images, to outline similarities and differences in performance.

A. Feature extraction algorithms

The four visual tasks implemented are described in the following subsections.

1) Edge detection: Edge detection has been implemented using the well known Canny edge detection operator. Given a grey scale picture, the image is first filtered with a Gaussian filter to remove noise. Then, a Sobel operator separates regions of high horizontal or vertical frequencies. Finally, the Canny operator is applied, leaving lines with a 1 pixel thickness, and a thresholding final pass provides a black and white image. Figure 2 illustrates these four steps.

2) Template matching: Template matching consists in finding whether (and where) a known given target template appears within a wider image. Template matching is very useful, for example, when beacons are scattered in the environment to help the robot recover from localization errors. For this operation, a simple template correlation was used. First, the two dimensional Fourier transform of the image is computed. Then the template image is transposed and padded to the size of the image. Next, the Fourier transform of the template is taken and multiplied with the transform of the image. The inverse transform of the result provides an image of the template convolution. We take the transpose of the template instead of the template itself because the algorithm needs to obtain the correlation of the two images and not the convolution. An example is show in figure 3. On the left is the template, followed by the inverted Sobel of the image and the final result. The darker regions are the locations in the image where the template is most probably located. In this example there are two distinct peaks, close to each other. Such variations occur when the size of the template does not exactly match that of the feature in the image, as this algorithm is not scale invariant. The usual practice to obtaining

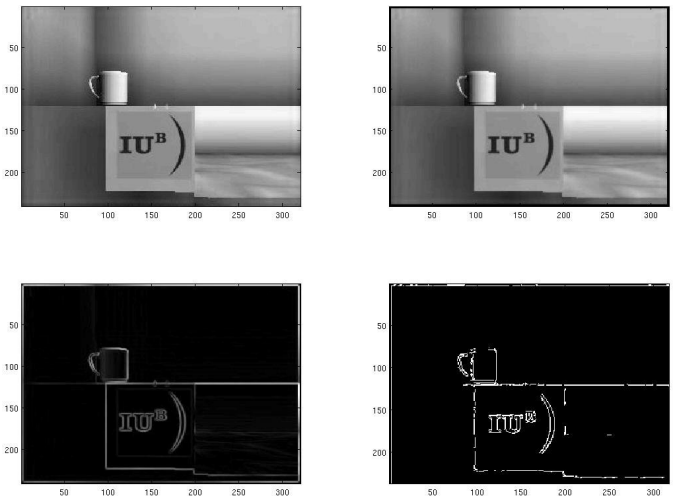

Fig. 2. The steps of the Canny edge detection operator

a scale invariant implementation involve generating a pyramid of possible templates of different sizes. A similar technique is used for obtaining rotation invariance, although in this case the problem is more complicated, due to the interpolation errors that occur when a digital image is rotated.

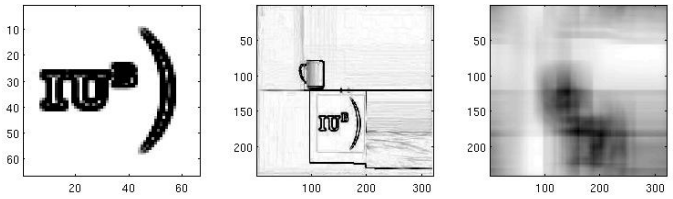

Fig. 3. Template matching. Picture of template (left), image with target feature(middle) and correlation(right)

3) Snakes: Active contours, also known as snakes, are one of the best performing feature extraction techniques available. The idea is the following: start with a number of points that encompass the target feature. The points form a contour with total energy

$$E_{snake} = \int_{s=0}^{1} E_{int}(v(\mathbf{v})) + E_{im}(v(\mathbf{v})) + E_{con}(v(\mathbf{v}))ds \quad (1)$$

where E_{int} is the internal energy of the contour, E_{im} is the energy component from the image and E_{con} is the constraint energy. The internal energy is implemented as the average distance between each two neighboring snake points, the constraint energy is the curvature of every three consecutive snake points and the image energy is proportional to the value of the pixel that the snake point is currently occupying. On each iteration of the algorithm the snake points are moved to minimize the snake energy and eventually shrink the contour to that of the targeted feature. There are several methods to solve rigorously and implement the continuous solution of the snakes algorithm in a discrete space. We have embraced the solution known as *the greedy snakes algorithm*, that performs a greedy search on points in the vicinity of each snake point. It computes a discretized version of equation 1 for each pixel in a 3 by 3 neighborhood and moves the snake point to the pixel that has the lowest value for E_{snake}. For the purposes of this algorithm the image energy is computed as the value of every pixel in a normalized, inverted Sobel edge transform of the original image. This implementation has a few inherent problems that sometimes lead to a complete failure of the algorithm. The first, and most serious shortcoming is that the snake points can get stuck at local minimums and stop moving. In general this is not that frequent, as if only one point moves this will likely trigger motion of other points and thus move the whole snake. To prevent cases when all points are stuck we have increased the size of the search window from 3 by 3 to 9 by 9 pixels, which has no considerable effect on the execution time, as the number of snake points is generally low. The second problem concerns the choice of weighting coefficients for each of the three components of the snake energy. Choosing a high value for the image energy makes snake points migrate to the closest edges and distort the original shape of the contour. Choosing too low a value on the other hand makes the contour static, because even small changes in the spacing between points and in the curvature have a huge impact on the total energy. Thus, choosing the proper constants becomes a tedious process that is specific for each image analyzed. Over a few tests constants that have a stable performance on the simulated images were chosen, again with the purpose of testing how well the tweaked algorithm would later perform on the real images.

4) Optical character recognition: Optical Character Recognition (OCR) is the problem of extracting text from raster images of text. There exist different algorithms to perform OCR. The one described here starts by properly aligning the text, so that all rows are parallel to the horizontal axis. This is achieved by computing the Hough transform of the text image and rotating it around an angle, equal to the most frequent Hough angle. Assuming we have a long enough text, all parallel lines that belong to characters will intersect in Hough space and thus the angle of rotation can be determined. The next step is to compute the vertical projection of the image and separate each element. This is possible, because of the white spaces between rows which are distinctly visible on the vertical projection of the image. Using a similar argument, we can compute the horizontal projection of each row and separate letters, also called *glyphs*. Individual letters are then cropped to ensure there are no extra white spaces. This algorithm is first performed on a learning image, which contains the whole character set to be recognized, in a known order. Thus, a database of characters and their respective glyphs is created. Subsequent text images are processed in the same way and for each character glyph a template matching is performed to find the character from the database that has the greatest similarity. An example is shown in figure 4: the orignal image, the image after thresholding and inverting, after rotation and after applying the noise reduction filter are displayed in sequence. This algorithm achieves a 100%

accuracy on images grabbed from the screen, but is susceptible to noise, as stray pixels, unless filtered, will be recognized as glyphs and matched against the database. The noise reduction filter was implemented to reduce salt and pepper noise and stray single pixels, but that has no effect on groups of noisy pixels. Filtering out such noise is very hard, as it is sometimes impossible to differentiate between a cluster of noisy pixels and a valid character.

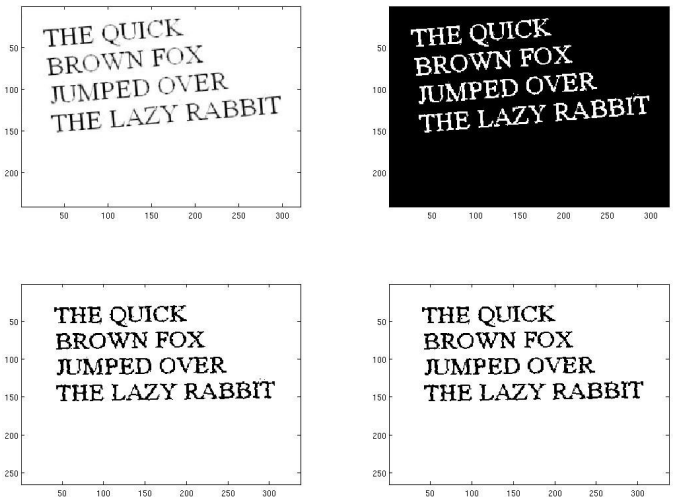

Fig. 4. The stages of Optical Character Recognition

All the above described algorithms have been implemented in Matlab closely following the descriptions found in [10] and [11].

B. Experimental setup and results

In order to compare the algorithm performance on corresponding simulated and real images, we have developed within USARSim a detailed model of a room environment and we have successively taken pictures from corresponding points of the virtual and real world. In order to test the algorithms under different boundary conditions, images with different light conditions were used.

Figure 5 presents the correlations between the edge images for eight test images. The autocorrelations of the simulated image in column 1 (dark blue) are comparable with those from the correlation between a well lit real world image and a simulated image (column 2, light blue). The same is true in most cases about the correlations of the simulation and the bad lit image, compared to the correlations of the well lit and bad lit image (columns 3 and 4, yellow and brown respectively). The slight deviations are mainly due to minor deviations of the positions of the camera when taking the images. It should be observed that the precise numerical value of the correlation is not the main aspect of this experiment. The relevant aspect is rather the gross scale similarity or discrepancy in the values.

Figure 6 presents the results for the distances (in pixels) between the actual position of the target feature (IUB logo displayed in figure 3 on the left) and the position estimated with template convolution. Except for the third and the seventh

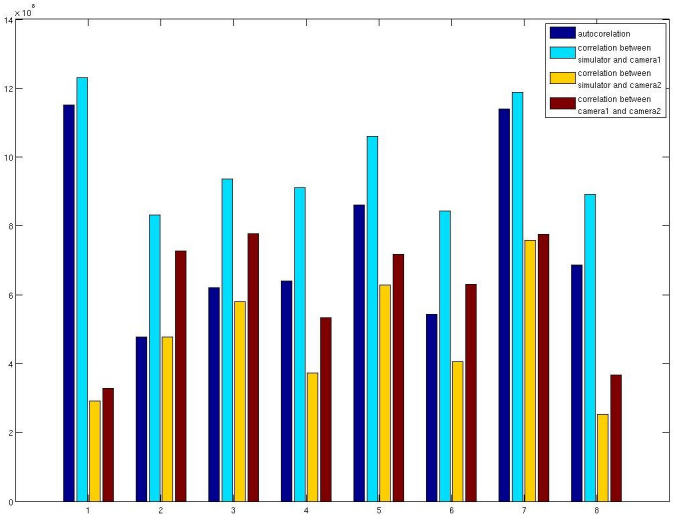

Fig. 5. Results for edge detection metric

image, the distances are below 100 pixels, which is about one and a half times the template size and a very good result. In most of the cases the results on the three images are very close, with the noticeable difference of image 6, where the well lit real image shows a much worse behavior than the other two. In most cases however, the performance is almost identical, as visual inspection of figure 7 (test image 1) confirms.

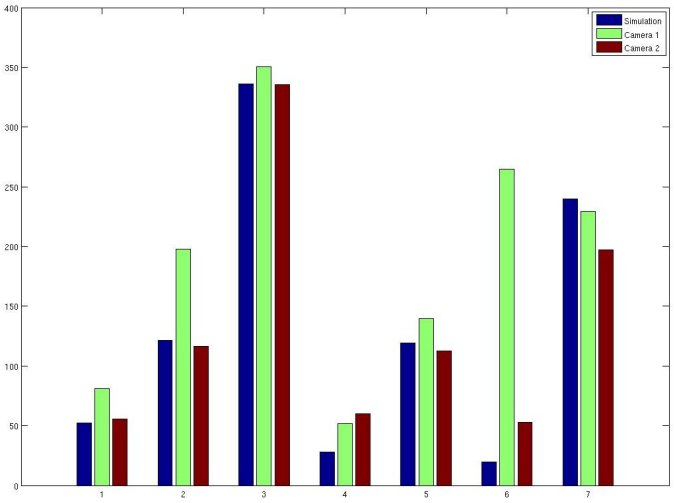

Fig. 6. Results for template convolution metric for simulation (blue), well lit conditions (green) and bad lit conditions (brown).

Figure 8 shows the average distance in pixels between snake points and target features for the three sets of images. The results show a maximum deviation of about 11 pixels, which is a good result, as well as some excellent performances on images 4 and 6 with average distance of about 3-4 pixels. The results for Image 6 are also presented in figure 9 (simulation) and figure 10 (real-world). Again, the performances on the three sets are comparable, and although the constants have been tweaked for the simulation, the real images sometimes outperform the simulated ones.

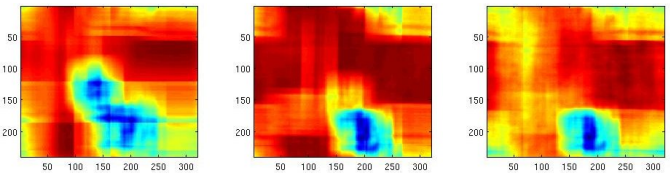

Fig. 7. Template convolution performed on simulator(left) and real world(right,middle)

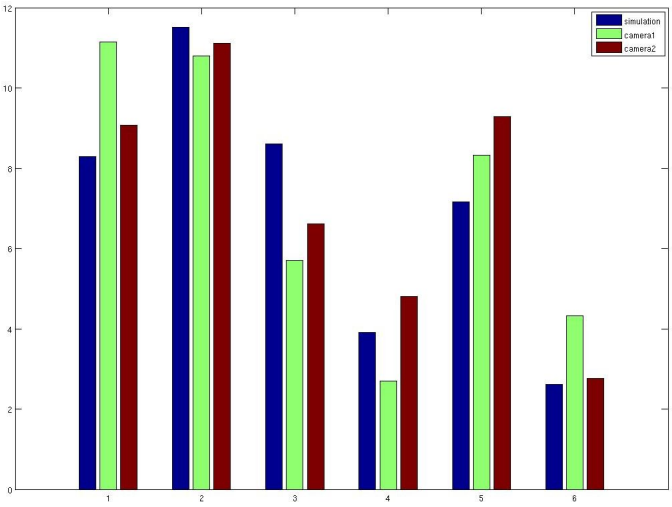

Fig. 8. results for the Active Contours metric

Finally, figure 11 presents the results of testing optical character recognition on two sets of images - one from the real model and one from the simulation. The figure measures roughly the percentage of recognized characters in each case. The success rate in both cases is pretty low, and noticeably lower in the case of the real world images. Inspecting the sample image in figure 12 gives a very good explanation for these low figures, i.e. the high level of noise. The figure shows the original images on the top - simulation on the left and real image on the right, as well as the images after filtering and rotation on the bottom. The bottom images exhibit a low quality and high fragmentation on the characters. This is due to the rigorous filtering that has removed most of the noise, but also parts of the characters. As the images from the real camera exhibit higher level of noise, they also have a lower quality after filtering and thus a lower success rate of recognition.

IV. Wireless simulation

An important factor in the performance of multi robot teams is the communication between the agents. In complex environments offering little or no opportunity for implicit information exchange, explicit communication can greatly improve the performance of multi-agent teams. USARSim currently does not provide any kind of simulation of communication mechanism, thus allowing all robots to freely communicate regardless of their position in the environment. To include a more realistic scenario in future USARSim releases, we have developed and validated a preliminary software module that mimics wireless

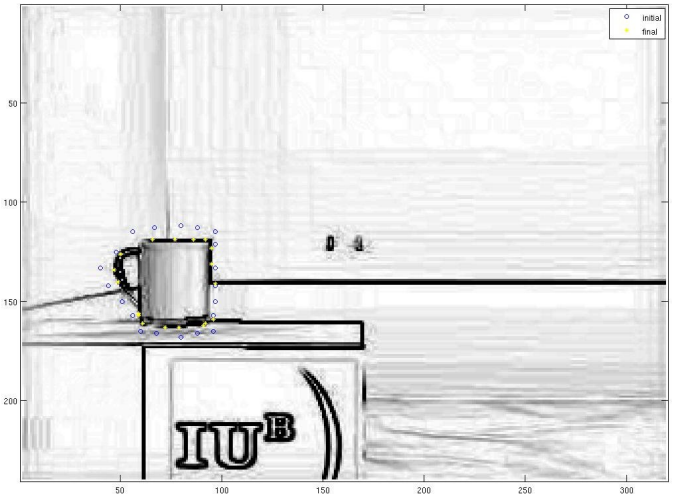

Fig. 9. Snake algorithm performed on simulated image

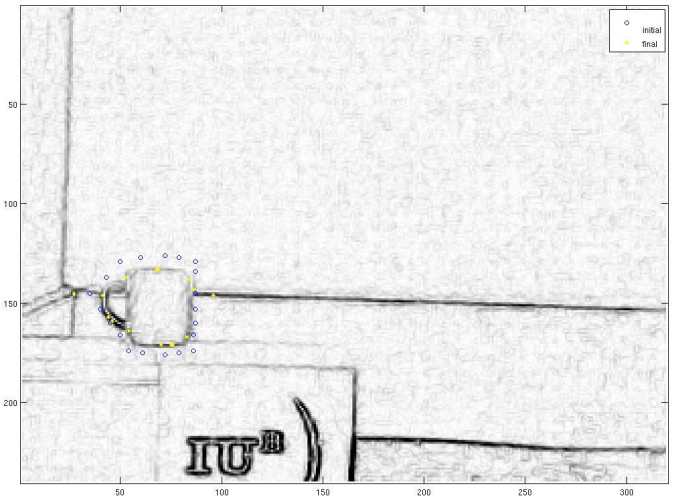

Fig. 10. Snake algorithm performed on real image

communication within simulated environments. As nowadays most robots use WaveLan cards to send messages to each other over wireless channels, the implementation of a WaveLan simulator for USARSim will greatly improve its accuracy as a tool to develop multi robot teams, hence making it even more attractive for the research community.

The simulation system consists of three modules. A so called *parser* component provides the infrastructure to compute the strength of a signal received by a receiver. A *server* component is used to dispatch messages from transmitters to receivers. Therefore if a the process controlling the simulated robot A desires to send a message to the process controlling the simulated robot B, it does not directly talk to it, but it rather asks the *server* to deliver a message. The server, upon inspection of the receiver signal strength, decides whether the message should be passed on or not. The third component, which will not be extensively described here, provides a one-to-one simulation of the socket API, so that communication software written within the simulator can be easily moved to

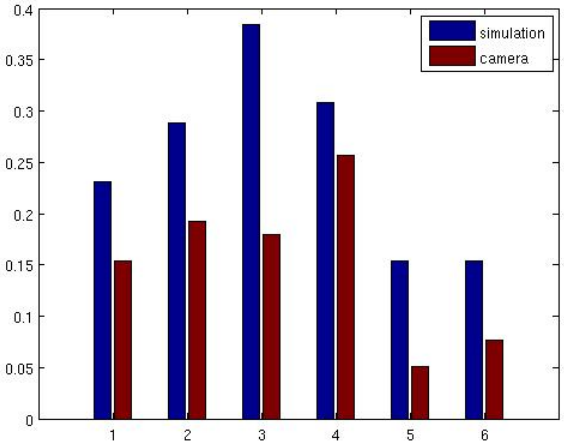

Fig. 11. Results of OCR metric

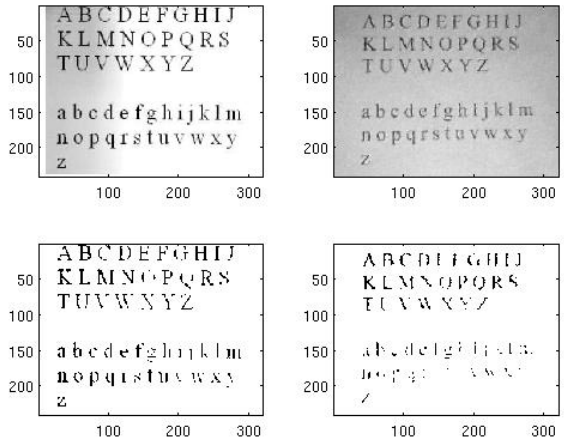

Fig. 12. OCR performed on simulator(left) and real world(right)

real robots.

A fundamental step for the simulation of wireless signals is the selection of a propagation model, i.e. a model describing how signals are propagated in the environment. Among the different ones proposed in the literature, we have selected the one presented in [12], also known as RADAR model. The model best predicts propagation within floors, accouting for the attenuation of the transmitted signal due to distance and traversed walls. The signal strength at a point at distance d from the emitter is modeled by the following equation

$$P(d) = P(d_0) - 10n \log\left(\frac{d}{d_0}\right) - \begin{cases} nW * WAF & nW < C \\ C * WAF & nW \geq C \end{cases} \quad (2)$$

$P(d_0)$ is the reference signal strength in dBm, nW is the number of obstructions between the transmitter and the receiver, and *WAF* is the so called *Wall Attenuation Factor*, i.e. an empirical factor accounting for the attenuation experienced by the signal while traversing a wall. C is the maximum number of obstructions up to which the attenuation factor affects the path loss. Finally, n is a factor indicating the rate with which the path loss increases with distance. It is therefore evident that if one wants to use equation 2 to predict the received signal strength, it is necessary to know the relative positions between the transmitter and the receiver, as well as the number of walls. This later number, needed to determine the right *nW* value, is not computed on the fly every time the value for $P(d)$ is needed, but is rather deducted from a data structure obtained by preprocessing once the map of the environment. The preprocessing operation is performed by the *parser* subsystem. According to the technical specifications of commercially available wireless devices the minimum receiver sensitivity is -92dBm. Therefore when the *server* receives a request for a message to be dispatched, it passes it on only if the received signal strength is above this value.

A. Testing and validation

In order to evaluate the performance of the proposed wireless simulation system we have developed within Unreal the model of an existing building. The environment features three fixed base stations that can be modeled as well within the proposed framework. A preliminary step has been the experimental determination of the parameters in equation 2. These values are displayed in table I.

Parameter	Value
Wall attenuation factor (*WAF*)	7
Maximum number of obstructions (C)	4
Path Loss factor (n)	1
Reference distance (d_0)	2
Signal strength at d_0 (dBm)	-50

TABLE I
EXPERIMENTALLY DETERMINED PARAMETERS

Next, for different placements of transmitters and receivers we have

- measured the actual signal strength in the environment
- computed the value predicted by equation 2
- computed the signal strength with the simulation system.

The results of these measurements and predictions are displayed in tables II, III and IV respectively.

Name	**No of Walls** [m]	**Average** [dBm]	**Median** [dBm]	**3rd Quartile** [dBm]
rtest1	1	-71.43	-71.2	-69.03
rtest2	1	-74.2	-74.05	-71.52
rtest3	0	-66.65	-67.04	-64.18
rtest4	2	-78.48	-77.7	-75.91

TABLE II
WIRELESS SIGNAL STRENGTH PREDICTED BY WAF MODEL

It can be observed that there is in general a good correspondence between the two predictions and the measured signals, although there are some obvious fluctuations. Large

Name	Average [dBm]	Std Dev [dBm]	Median [dBm]	3rd Quartile [dBm]
rtest1	-72.18	6.37	-68	-67
rtest2	-70.85	2.47	-71	-70
rtest3	-66.97	7.44	-63.5	-61
rtest4	-73.95	1.34	-74	-73

TABLE III
EXPERIMENTAL VALUES FOR WIRELESS SIGNAL STRENGTH

Name	Average [dBm]	Median [dBm]	3rd Quartile [dBm]
rtest1	-71.33	-71.09	-68.93
rtest2	-81.12	-80.42	-78.29
rtest3	-73.76	-73.6	-71.12
rtest4	-78.25	-77.46	-75.71

TABLE IV
WIRELESS SIGNAL STRENGTH VALUES FROM THE SIMULATOR

discrepancies between the results predicted by the simulator and those forecasted by the RADAR module are explained by the approximations introduced by the *parser* module.

V. HUMAN ROBOT INTERACTION

Validating USARsim for human-robot interaction (HRI) presents a complex problem because the performance of the human-robot system is jointly determined by the robot, the environment, the automation, and the interface. Because only the robot and its environment are officially part of the simulation, validation is necessarily limited to some particular definition of interface and automation. If, for example, sensor-based drift in estimation of yaw were poorly modeled it would not be apparent in validation using teleoperation yet could still produce highly discrepant results for a more automated control regime. Our validation efforts for HRI, therefore, sample two widely used control schemes [13], teleoperation and point-to-point control for two robots, the experimental PER [14] and the commercial Pioneer P2-AT (simulation)/P3-AT (robot) in order to provide an indication of the likely validity of the simulation for HRI across a range of configurations.

We have completed validation testing at Carnegie Mellon's replica of the NIST Orange Arena for the PER robot using both point-to-point and teleoperation control modes reported in [15] and have collected teleoperation data for the Pioneer reported in [3]. In these tests robots were run along a narrow corridor in either the simulation or the Orange Arena with three types of debris (wood floor, scattered papers, lava rocks) while the sequence, timing and magnitude of commands were recorded. In the first three trials, participants had to drive approximately three-meters, along an unobstructed path to an orange traffic cone. In the next three trials, obstacles were added to the environments, forcing the driver to negotiate at least three turns to reach the cone yielding a between groups design pairing each surface type with straight and complex paths.

The paper surface had little effect on either robot's operation. The rocky surface by contrast had a considerable impact, including a loss of traction and deflection of the robot. This was reflected by increases in the odometry and number of turn commands issued by the operators even for the straight course. A parallel spike in these metrics is recorded in the simulator data. As expected the complex course also led to more turning even on the wood floor. Figure 13 shows task times for real and simulated robots. Differences within conditions were low particularly for complex paths which are more likely to be influenced by human control suggesting that USARSim is likely to provide a valid tool for investigating HRI.

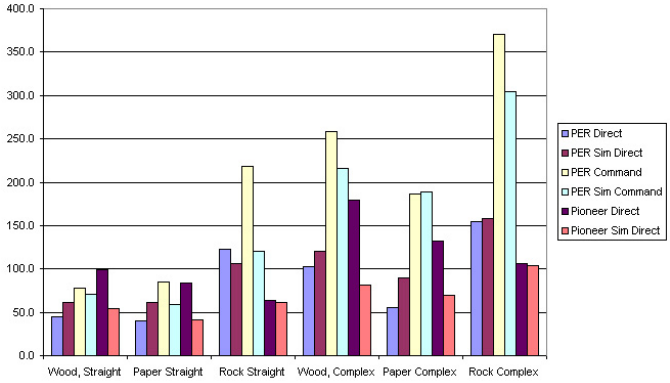

Fig. 13. Task Duration

The one metric on which the simulation and the physical robot consistently differed was proximity to the cone when teleoperating the PER (14). Operators using the physical robot reliably moved the robot to within 35cm from the cone, while the USARSim operators were usually closer to 80cm from the cone. It is unlikely that the simulation would have elicited more caution from the operators, so this result suggests that there could be a systematic distortion in depth perception, situation awareness, or strategy.

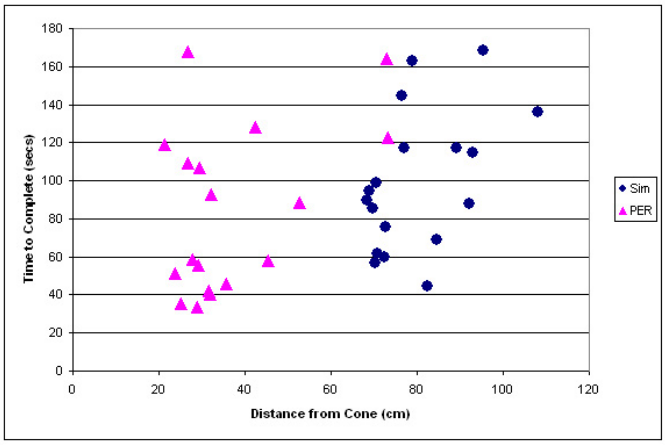

Fig. 14. Approach to Cone for Teleoperated PER

VI. Conclusions

This paper describes validation tests for feature extraction from simulated images, a radio propagation model, and tests involving human control. The feature extraction tests are especially important to validating the simulator because of the complexity of the visual imagery. The underlying game engine was explicitly designed to generate imagery that would appear realistic to human perception. This is, however, no guarantee that the information extracted from synthetic images would correspond to that extracted from real camera views. In fact, the clarity and lack of naturally occurring distortion in synthetic images might be expected to yield perfectly formed extractions where nothing might be found even in clear appearing real images. Our results are very encouraging because they show a close correspondence between information extracted from real and computer generated images at least under well lit conditions. Further validation will be required to determine whether this correspondence will extend to other illumination levels and extraction algorithms. The radio simulation, by contrast, provides a validated tool for approximating communications difficulties at USAR tasks for use with the simulator but does not reflect on the validity of the simulator itself. The driving tests showed that robots in simulation behaved in much the same way as real robots. The correspondence in performance for robots and simulation between control modes, terrain type, and task complexity suggest that the simulation is both physically accurate and presents similar challenges to human operators making it an appropriate tool for HRI research.

To draw valid conclusions from robotic simulations it is important to know the metrics which are consistent with the operation of the actual robot and those which are not. By collecting validation data for all entities within the simulation we hope to create a tool with which researchers can pick and choose manipulations and metrics that are likely to yield useful results. As our library of models and validation data expands we hope to begin incorporating more rugged and realistic robots, tasks and environments. Accurate modeling tracked robots which will be made possible by the release of UnrealEngine3 would be a major step in this direction.

References

[1] J. Wang, M. Lewis, and J. Gennari, "Usar: A game-based simulation for teleoperation," in *Proceedings of the IEEE International conference on systems, man and cybernatics*, 2003, pp. 493–497.

[2] S. Carpin, J. Wang, M. Lewis, A. Birk, and A. Jacoff, "High fidelity tools for rescue robotics: results and perspectives," in *Robocup 2005: Robot Soccer World Cup IX*, ser. LNCS, 2006, pp. 301–311.

[3] S. Carpin, M. Lewis, J. Wang, S. Balakirski, and C. Scrapper, "Bridging the gap between simulation and reality in urban search and rescue," in *Robocup 2006: Robot Soccer World Cup X*, ser. LNCS.

[4] R. Vaughan, B. Gerkey, and A. Howard, "On device abstractions for portable, reusable robot code," in *Proceedings of the IEEE/RSJ IROS*, 2003, pp. 2421–2427.

[5] "Player/stage project," http://playerstage.sourceforge.net, 2005.

[6] R. Brooks, "A robust layered control systems for mobile robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, pp. 14–23, 1986.

[7] ——, "Intelligence without reason," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1991, pp. 569–595.

[8] A. Jacoff, E. Messina, and J. Evans, "Experiences in deploying test arenas for autonomous mobile robots," in *Proceedings of the 2001 Performance Metrics for Intelligent Systems (PerMIS)*, 2001.

[9] Mathengine, *Karma User Guide*. [Online]. Available: http://udn.epicgames.com/Two/KarmaReference/KarmaUserGuide.pdf

[10] M. Nixon and A. Aguado, *Feature extraction and image processing*. Newnes press, 2002.

[11] J. Parker, *Algorithms for image processing and computer vision*. Wiley Computer Publishing, 1997.

[12] P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *INFOCOM (2)*, 2000, pp. 775–784.

[13] J. Crandall, M. Goodrich, D. Olsen, and C. Nielsen, "Validating human-robot interaction schemes in multi-tasking environments," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, no. 33(3), pp. 325–336, 2003.

[14] I. Nourbakhsh, E. Hamner, E. Porter, B. Dunlavey, E. Ayoob, T. Hsiu, M. Lotter, and S. Shelly, "The design of a highly reliable robot for unmediated museum interaction," in *2005 IEEE International Conference on Robotics and Automation (ICRA'05)*, 2005.

[15] J. Wang, M. Lewis, S. Hughes, M. Koes, and S. Carpin, "Validating usarsim for use in hri research," in *Proceedings of the Human Factors and Ergonomics Society 49th Annual Meeting (HFES'05)*, 2005, pp. 457–461.