High-Fidelity Simulation for Robotic Vision Research

John Skinner[1], Sourav Garg[1], Niko Sunderhauf[1], Peter Corke[1], Ben Upcroft[1], Michael Milford[1]

***Abstract*— One of the critical bottlenecks of computer and robotic vision research has been the extent to which real world image data could be captured for both training and testing purposes. In domains such as robot navigation, certain types of highly relevant data is difficult and labor intensive to obtain - for example, multiple traverses of a road under all possible environmental conditions or from all possible camera viewpoints. In this paper, we investigate how these limitations can be addressed by using a high-fidelity simulation environment to generate image datasets. To achieve this, we use an existing state of the art simulation tool, Unreal Engine 4 by Epic Games, to generate a variety of image datasets that contain task-relevant variation including camera-viewpoint change and time of day changes, and demonstrate that algorithm performance in the real world is similar to in simulation. These datasets enable us to systematically evaluate current state-of-the-art place recognition algorithms at a granularity that was previously impossible, both in terms of comparing algorithms to each other but also evaluating the effect of varying key algorithm parameters. Together, this work opens up new opportunities for development and analysis of algorithms, without the added cost of capturing real-world datasets with which to test the algorithm under development.**

I. INTRODUCTION

Computer vision research typically relies on image datasets collected from the real world to test, train and analyse the algorithms developed. But acquiring the necessary image data from the real world can be difficult, expensive or, for certain kinds of data, outright impossible. For instance, research on algorithms that desire to be viewpoint invariant or condition invariant requires images from a wide variety of viewpoints or a variety of conditions.

Another key limitation on real-world image datasets is a lack of repeatability. Usually each experimental run or dataset capture is unique, as changes in time of day or time of year or weather all change lighting conditions, sky and, background elements. Even the exact position of the camera will vary at the small scale, shifting pixels and producing similar but distinct images, even in the most static environments.

It is also often difficult to capture a sufficient amount of data from the real world. When working with supervised learning algorithms, such as modern deep-learning neural nets, a large amount of data is required to train and test the network. Crucially, this data must be of sufficient variety that the resulting network can generalize successfully to unseen data rather than over-fitting to the training data.

Further, a reliance on limited real-world datasets in turn limits the ability to demonstrate the generality and limitations of an algorithm. Take for instance the demonstrations of depth invariance in SMART [8], in which the adjusted S-MART algorithm is shown to be capable of identifying places given a 4-lane change in depth. No claims are or can be made about the limitations of this technique; over how large a depth-change can the algorithm maintain performance? 100m? 1 km? How does the performance fall off as the depth increases? We cannot answer these questions using real-world data, as collecting the requisite images would be impossible.

In this paper, we demonstrate that many of these issues can be addressed by generating image data on demand from a high-fidelity simulation environment. High-fidelity simulation is capable of generating arbitrary amounts of image data, which can be used to both train and test computer vision algorithms.

II. PRIOR RESEARCH

A. Simulation

The use of simulation in robotics is not a new idea. Popular robot simulators such as Gazebo [4] and Player/Stage [2] have existed for many years, and are commonly used to test robot motion and control. The use of game engines for robot simulation is also not new, with the USARsim project based on the Unreal Tournament 3 engine [1], and other projects using the Unity engine [5]. Nobody has yet made use of modern, high-fidelity engines such as Unity 5 or Unreal Engine 4 (the successor to the Unreal Tournament 3 engine).

The more specific field of robotic vision has not seen much application of simulation at all, it instead prefers to work from image datasets captured from the real world. The key requirement for simulation in computer vision is that the simulator is capable of photo-realistic rendering and lighting, which has historically been out of reach for older platforms, including the Unreal Tournament 3 engine or Gazebo. Unreal Engine 4 however has powerful tools for realistic materials and lighting [3], which make it possible to create simulated environment that produce images similar to the real world.

Unreal Engine 4 has a number of additional advantages that make it suitable as a simulator platform for computer vision. It is developed and maintained by Epic Games Inc, and uses physics and modelling tools from nVidia, which allow for complex and interactive dynamic environments. It also allows full access to it's source code, allowing a stimulation designer complete control over the simulation. It is also free for non-commercial uses, including research.

B. Place Recognition

- Sequence SLAM

[1] Australian Centre for Robotic Vision, Queensland University of Technology, 2 George St, Brisbane, Australia

  - Developed by Milford and Wyeth... (citation)
  - Performs place recognition using
- Challenges
  - challenges include viewpoint invariant place recognition, condition invariant place recognition, place recognition in dynamic environments
  -

III. ANALYSIS OF PLACE RECOGNITION

To demonstrate the effectiveness of high-fidelity simulation as an analysis tool for computer vision, we performed an in depth analysis of the viewpoint and time-of-day invariance of two different state-of-the-art place recognition algorithms, SeqSLAM [6] and OrbSLAM [7]. Then, to demonstrate the versatility of we performed an analysis of the viewpoint dependence of object recognition.

A. Simulation Setup: Unreal Engine 4

The simulation tool used in this paper was the Unreal Engine 4, developed by Epic Games Inc.

All of the test images used were generated from the same street scene, shown in Figure 1. The 3D models used were either sourced for free from TurboSquid (www.turbosquid.com), with significant manual clean up, or were produced manually. The landscape was produced using the basic version of WorldMachine (http://www.worldmachine.com). All lighting is as computed by the Unreal Engine, using standard sky and light assets provided with the engine.

To generate the image data, the camera was made to move along a specified path and produce images at fixed intervals. The simulator allows us to precisely repeat the same path, and introduce calculated and precise variations upon it. In order to explore how the performance of place recognition changes with time of day and viewpoint change, passes along the path were generated for 5 different times of day in combination with increasing lateral offsets or camera tilts (Figure 6).

B. Place Recognition: Sum of Absolute Differences

The first test we did was on simple place recognition using simple sum-of-absolute-differences (SAD) based image matching. While this is neither state-of-the-art nor particularly effective, sum-of-absolute-differences is a common metric which is used in many computer vision algorithms, including SeqSLAM discussed below. As such, an analysis of it's performance characteristics is potentially interesting.

Before matching, each image is down-sampled to 64x64 pixels and reduced to greyscale, in order to save computation time. The matcher used compares each image in a query dataset to each of those in a reference dataset, and considers the reference image with the lowest sum of absolute difference to be a match. The performance is then the percentage of images for which the matched reference image is taken from a place close to the query image.

The matcher is tested using 130 different combinations of time of day and viewpoint changes, so that we can see how the performance falls off as both increase. The results are summarized in Figure 7.

Figure 7 shows that SAD-based matching seems to be relatively robust against small lateral offsets, falling of after approximately $2m$. There is similar falloff as time of day approaches sunrise or sunset, and seems to be relatively symmetrical. The small number of samples makes this change seem relatively smooth, but it also seems plausible that it may instead change rapidly around dawn and sunset when the lighting change has the most effect. Resolving this is simply a matter of sampling the distribution further.

Interestingly, matching rate falls off similarly with angle change irrespective of the direction of change (compare the lower two plots in Figure 7). Matching performance seems to follow an exponential decay with angle, but additional sampling would need to be performed to verify that hypothesis. It may also be worth investigating the effects of multiple orientation offsets combined together to see how they compound. Were it the aim of this paper, it would be relatively simple to generate additional required sample passes to properly evaluate, but it is beyond the current scope.

C. Place Recognition: SeqSLAM

The first full place recognition algorithm we investigated was SeqSLAM, first described by Milford and Wyeth [6]. SeqSLAM is a place recognition algorithm that searches for loop closures by attempting to match sequences of similar images. It measures image similarity using sum of absolute differences as analysed in the previous section.

To perform the test, we generated 130 passes across 5 times of day and 26 different viewpoint variations from the street scene used in the other tests (figure 6. We chose the baseline pass at noon as the reference dataset, and compared it to all 130 of the other datasets (including itself). For each dataset we calculated the maximum F1 score, the results are summarized in Figure 8.

The first immediately obvious result is the anomalous low performance values for a left 0.2m offset in the morning and at sunset, and the low performance across all times of day at a 10°vertical orientation change. All of these changes are sudden and extreme, so the initial inclination is to sample around these points to see if there is a smooth or sudden decline. Based on the results of tweaking the parameters however, it seems more likely that SeqSLAM performance is sensitive to having achieved a certain level of matching, and that it will either perform very well or badly, rather than smoothly transition.

The other feature of note in the performance characteristics is the way the F1 score plateaus for translational offsets of less than 1 and orientation changes of 5°. When the algorithm performs well, it seems to do so irrespective of the time of day. Strong condition invariance has been noted in previous work as a particular feature of SeqSLAM [6], so it is good to see it confirmed here. Note however that when the performance falls off, it becomes less condition invariant, falling off further toward sunset and sunrise.

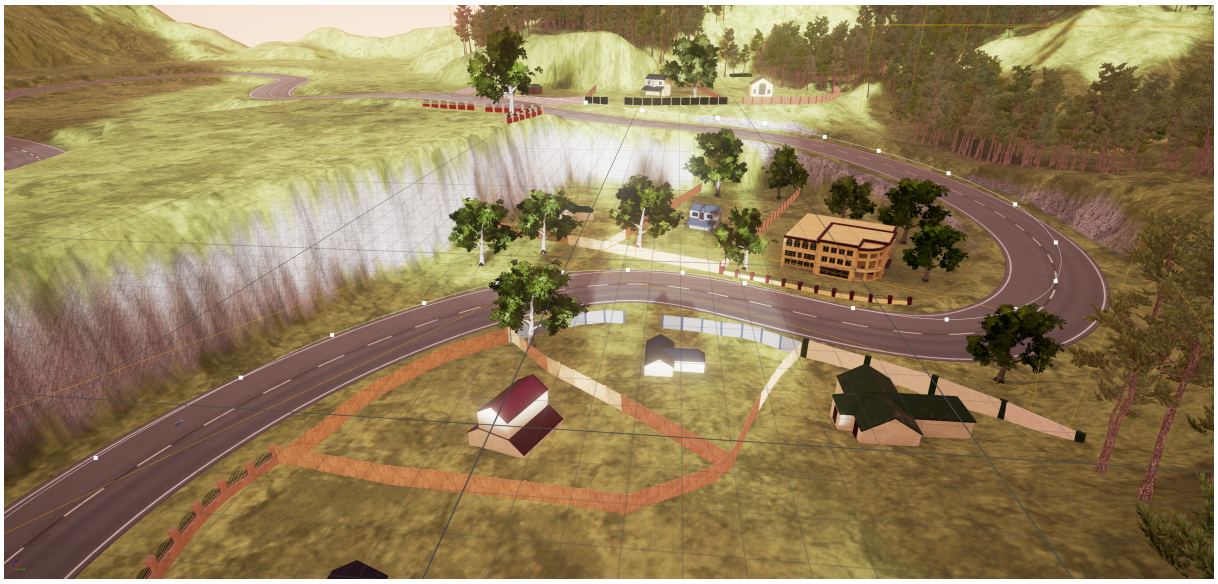

Fig. 1. The street scene used to capture the image datasets that were used for testing the place recognition algorithms. The line of white dots shows the baseline path followed by the camera when generating the datasets to test Sum of Absolute Differences matching and SeqSLAM. OrbSLAM test datasets follow a slightly different path so that they can loop their way around and end where they started.

Fig. 2. Sample images used to test lateral offset viewpoint invariance. From left to right, images are offset are left $3.5m$, $2m$, $1m$, $0.4m$, $0.2m$, Then the baseline, then offset right $0.2m$, $0.4m$, $1m$, $2m$, and $3.5m$

Fig. 3. Sample images used to test orientation viewpoint invariance. From left to right, images are angled are up $30°$, $15°$, $10°$, $5°$, Then the baseline, then angled down $5°$, $10°$, $15°$, and $30°$

Fig. 4. Sample images used to test orientation viewpoint invariance. From left to right, images are angled are left $30°$, $15°$, $10°$, $5°$, Then the baseline, then angled right$5°$, $10°$, $15°$, and $30°$

D. Place Recognition: OrbSLAM

The second place recognition algorithm tested was OrbSLAM [7]. OrbSLAM is a feature-based monocular SLAM system, that performs feature-based visual tracking, mapping and loop closure using ORB features.

To test OrbSLAM, we again generated a variety of image datasets from the same street scene, with 5 different times of day, a baseline pass and 20 different viewpoint variations. Due to the mapping element of OrbSLAM, all datasets were made to start and end in the same place. To test the performance, each of the datasets was appended to the noon baseline dataset (acting as a reference pass), and we looked for loop closures between the two passes. Results are summarised in Figure 9.

E. SeqSLAM parameter tuning

The detailed analysis we have performed can be used to inform subsequent actions, including parameter tuning. To demonstrate this, we again tested the performance of Sum of Absolute Differences on the noon datasets across the range of lateral offsets shown in Figure 2 with a variety of different offset window parameters. SeqSLAM uses Sum of Absolute Differences combined with an offset window parameter that shifts the image pixels to achieve a better matching image.

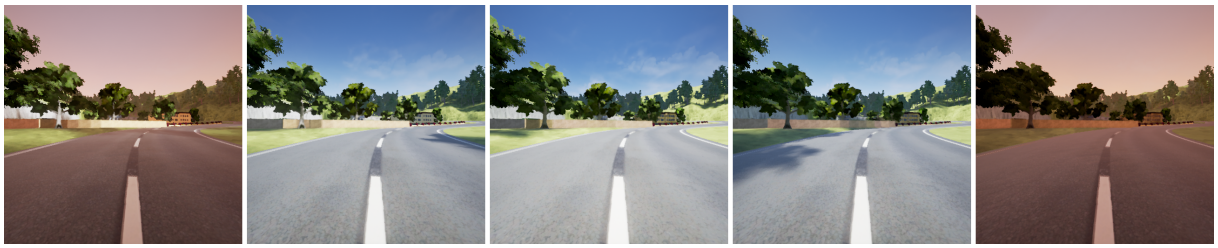

Fig. 5. Sample images used to test condition invariance. From left to right, images are taken at dawn, in the morning, at noon, in the afternoon, and at sunset

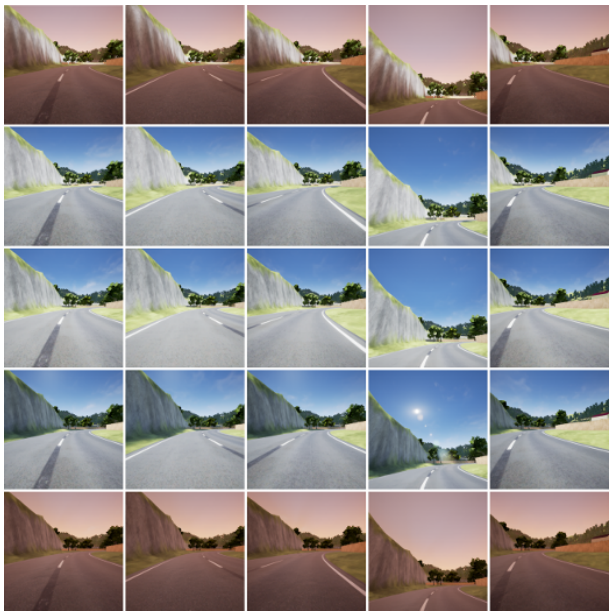

Fig. 6. Sample images from the datasets used, all images are taken from the same place at different times of day and viewpoint variations. From left to right, viewpoints are the base unmodified path, offset left 2m, offset right 2m, angled up 30°, and angled right 30°

The results of this test can be seen in Figure 10.

This data suggests that for distances up to 3.5m offset from the reference location, it is best to use the smaller 4 pixel window range. The use of an offset window shows clear improvement in performance, but larger windows may introduce false matches. Larger windows may be appropriate to larger offset, and if necessary we could easily have tested this by generating more data and extending the experiment.

F. Comparison to real-world

It is important to verify that performance change in simulation is representative of performance change in the real world. To test this, we compared performance drop over lateral viewpoint change using SeqSLAM for data from a real street and for data generated from a simulation of a similar street. The results can be seen in Figure 11.

Unsurprisingly, observed performance is consistently better in simulation; we ascribe this to a lack of sufficient realism in the simulation, which was built simply and quickly. However, the performance follows the same trends in both environments, falling off smoothly as the offset difference increases. This demonstrates that performance results obtained in simulation obey similar trends to data obtained from even simplistic simulations.

IV. DISCUSSION

The most difficult and time consuming aspect of using high-fidelity simulation is creating the scene in the first place. Scenes are built of 3d models and textures, which are first time consuming to create, and then take even more time to arrange in the scene. The more realistic a scene is required to be, the longer both of these stages take. The creation of high quality 3D models and scenes is the speciality of a professional 3D artist, so if a particular problem requires large or realistic scene it may be necessary to hire a 3D artist to construct it. It can also be extremely difficult to maintain a consistent scale throughout the scene. Examination of our scene in Figure 1 will reveal a myriad of scale flaws, note for instance the height of the houses relative to the width of the road.

However, once a particular simulation has been created, it can be used to generate arbitrary amounts of image data. Once we had constructed the street scene and the tools set up, actually capturing each dataset can be specified programatically and takes relatively little time. Indeed, when we initially created the datasets used to test the place recognition algorithms above, we tested orientation change at $30°$ and $15°$ only. However, after observing the way matching performance falls off with angle in Figure 7, we were able to very easily add tests for $5°$ and $10°$ orientation changes as well. The very fact that we can exactly repeat a movement through a scene with a precise orientation change is a powerful advantage of high-fidelity simulation.

It can in some sense be too easy to capture data. Since capturing additional data is often simply a matter of adding a new modifier to a path, increasing a sample range or decreasing a sample increment, it can be very easy to generate too much data. For instance, it may be tempting to sample a street scene such as ours at every offset up to 4m either side in 1m increments, and at each location take all vertical and horizontal changes up to $30°$ in $5°$ intervals. This is relatively simple to specify, but when multiplied out, produces $9 \times 13 \times 13 = 1521$ images per forward step down the path. The path we used with a step distance of 1m as well requires 612 forward steps, for a total therefore of 1965132 images. When capturing datasets ourselves, we

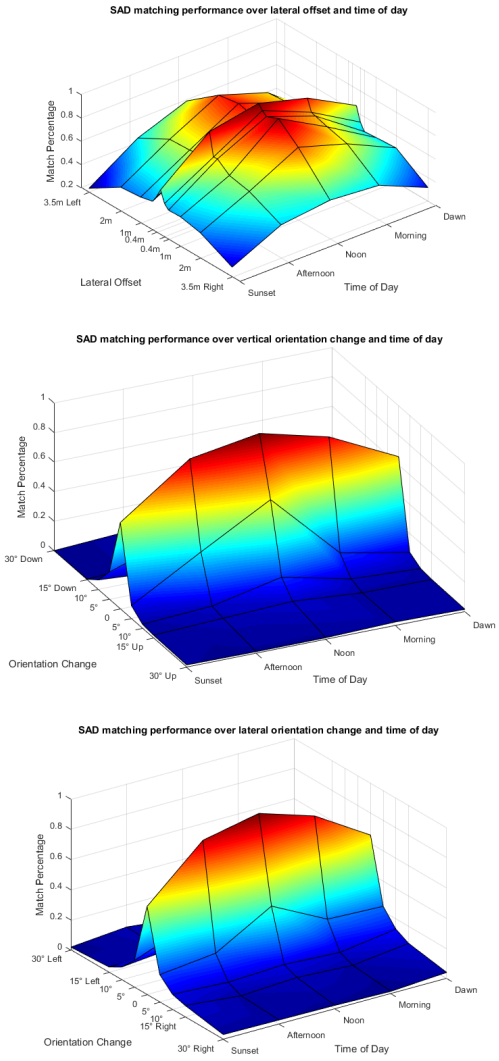

Fig. 7. Match percentage for Sum of Absolute Differences for, from top to bottom, lateral offset, vertical orientation change (pitch), and lateral orientation change (yaw). Each offset was tested across several times of day.

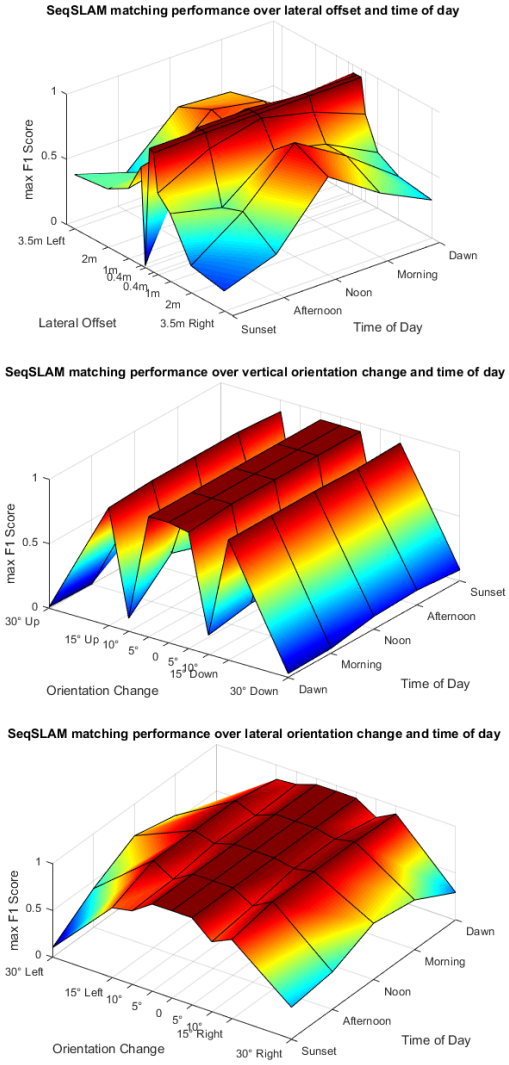

Fig. 8. SeqSLAM performance over, from top to bottom, lateral offset, vertical orientation change (pitch), and horizontal orientation change (yaw). Each viewpoint change is captured over 5 times of day to show how performance degrades over both viewpoint and condition change

averaged about 6 frames per second, so collecting all this data would take approximately 91 hours. It's easy to see how tweaking any of the specified numbers could multiply this number even further.

Rather than sampling densely with a large initial dataset, we recommend initial testing be done relatively sparsely over the test domain, and then using the initial results to choose a second round of test values. For instance, given the distribution for vertical orientation change in Figure 7, it makes more sense to generate new test data at $2.5°$ and at $7.5°$ than at $25°$. In the future, it should be possible to automate this iterative testing, automatically choosing new test datasets based on the results of previous testing.

It is also important to note that the difficulty of a particular change to the simulation can be non-intuitive. For instance, it is very easy in the simulation to change the location or orientation of an object or the camera; or to change the base colour of a flat-coloured object. For this reason, it was very easy for us to add additional variations on the camera path, since this simply changes the camera's location and orientation. On the other hand, changing the lighting is simple manually, but for good quality lighting a lot of data needs to be recalculated, which takes a lot of time and is not designed to be triggered programatically. As such, it takes longer to generate data across different times of day, and it would be more time-consuming for us to test at an additional time of day.

A. *Future Work*

There remains a significant amount of applications that we have not yet explored. At this time, we have not successfully tested interaction between robot control code and high-fidelity simulation, further work is required on the code interfaces that make this possible. With further investigation, it should be possible to create full simulators for computer

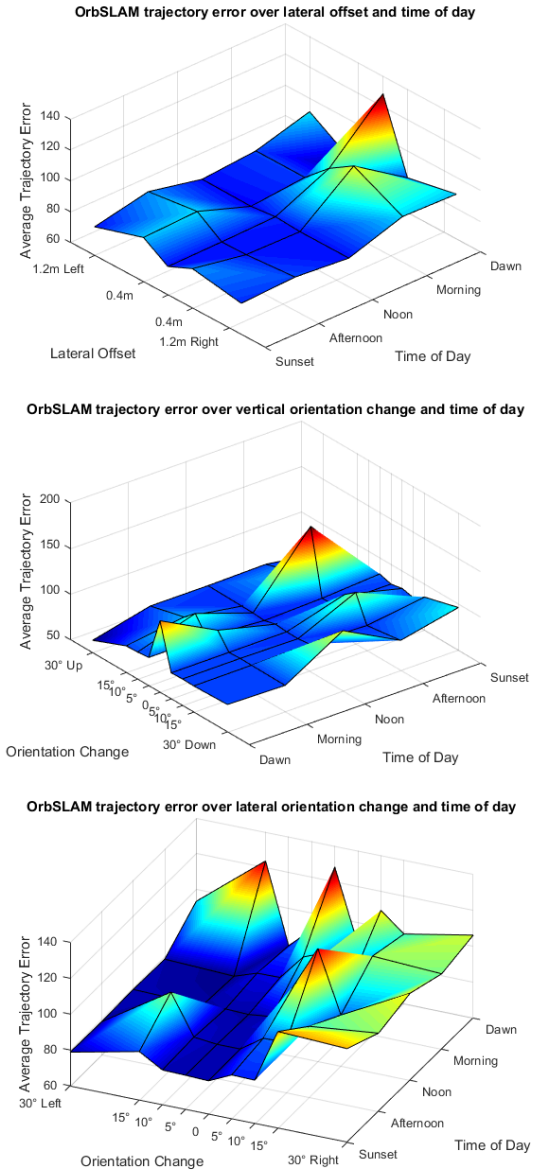

Fig. 9. OrbSLAM average trajectory error over, from top to bottom, lateral offset, vertical orientation change (pitch), and horizontal orientation change (yaw). Each viewpoint change is captured for 5 different times of day.

vision code including control code that interacts with the simulated world. This opens up the advantages of high-fidelity simulation to all computer vision algorithms, rather than only to passive algorithms.

This paper has focused on using simulation to test and analyse computer vision algorithms, but the generation of image data can also be used to train learning algorithms, such as deep-learning neural nets. Interestingly, it should be possible to apply the iterative testing approach outlined here to network training, alternating between generating new test data to fill holes in the performance graph and generating new training data to correct areas of poor performance, with a reduced risk of over-training.

The key problem with using high-fidelity simulation is that

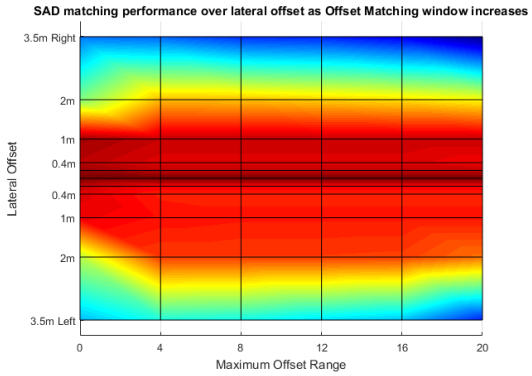

Fig. 10. Sum of Absolute Difference matching rate as the maximum offset window is increased. This result helps us choose a maximum offset appropriate for a given lateral offset

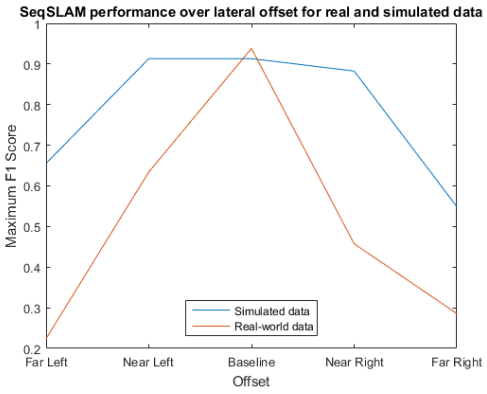

Fig. 11. Comparison of SeqSLAM performance falloff between simulated and real-world environments as lateral offset increases. As is often the case, simulated performance is better in an absolute sense, but the trend is the same in both cases.

generating simulation environments is too costly. There are a number of different ways that this can be made easier. Firstly, it is often desirable for simulation environments to mirror a real-world environment in some way, and so it should be possible to use modern 3D reconstruction techniques to lay the groundwork for a simulation environment. It should also be possible to procedurally generate more about the environment, establishing rules for the generation of roads, cities, and other environments allowing variations to be produced quickly. Finally, many assets need only be created, and a wider community using simulation and sharing work should reduce the initial costs of setting up a simulation.

APPENDIX

Appendixes should appear before the acknowledgment.

ACKNOWLEDGMENT

REFERENCES

[1] Stefano Carpin, Mike Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. USARSim: A robot simulator for research and education, 2007.

[2] Brian P Gerkey, Richard T Vaughan, and Andrew Howard. The Player / Stage Project : Tools for Multi-Robot and Distributed Sensor Systems. *Proceedings of the International Conference on Advanced Robotics (ICAR 2003)*, (Icar):317–323, 2003.

[3] Brian Karis and Epic Games. Real shading in unreal engine 4. *part of Physically Based Shading in Theory and Practice, SIGGRAPH*, 2013.

[4] N. Koenig and a. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 3:2149–2154, 2004.

[5] William A Mattingly, Dar-jen Chang, Richard Paris, Neil Smith, John Blevins, and Ming Ouyang. Robot design using unity for computer games and robotic simulations. In *2012 17th International Conference on Computer Games (CGAMES)*, pages 56–59. IEEE, 2012.

[6] Michael J. Milford and Gordon F. Wyeth. SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1643–1649, 2012.

[7] Raul Mur-Artal, JMM Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *Robotics, IEEE Transactions on*, 31(5):1147–1163, 2015.

[8] Edward Pepperell, Peter I Corke, and Michael J Milford. Automatic image scaling for place recognition in changing environments. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1118–1124. IEEE, 2015.