Construction and Validation of a High Fidelity Simulator for a Planar Range Sensor

Abhijeet Tallavajhula
atallav1@andrew.cmu.edu
Carnegie Mellon University

Alonzo Kelly
alonzo@cmu.edu
Carnegie Mellon University

***Abstract*— We study the problem of building a sensor model for the purpose of simulation. Our work is motivated by the potential impact of realistic simulators on the development cycle of software for real robots. The case is made for building models from approximate state information, relieving the burden of ground truth. Unlike calibration, where the goal is to identify and remove error from a signal, our aim to reproduce the signal in its entirety, including its error properties. Instead of physically modeling sensor behavior, a data-driven approach is taken. The implementation of our approach to simulate a simple but noisy laser rangefinder is described. Finally, approaches to validate the simulator are discussed. We compare not only raw sensor predictions, but also overall performance of algorithms on simulated versus real data.**

I. Introduction

Perception systems that operate on data from rangefinders, such as [16], need to be robust to noise in the sensors and the real world. The algorithms in the pipeline often preprocess the data and have parameters for thresholds such as outlier checks. We are motivated in this paper by a desire to reduce the cost and complexity of robot algorithm development while increasing the quality of the resulting software. The conventional wisdom is that software developed in simulation will need significant rework before it also works on hardware. We try to adopt a perspective that if simulators were only good enough, this would no longer be the case. Indeed, if the data produced by a simulator were as poor in quality as real data, then software developers could address the right issues earlier in the development cycle, do so using better debugging tools, and produce a higher quality result with less effort. Existing simulators are often not sufficient to meet this demand. They add simple identical noise to rays [10], and in some cases support finer-grained noise with more adjustable parameters [6]. There is little guidance provided as to what the noise settings must be, or how to determine them. We are not only interested in an accurate sensor model, but also a simple and principled procedure to acquire the model, which users can implement for their own sensors.

Since our work is about acquiring a sensor model and using it for simulation, an obvious related field is calibration. An old problem that has been repeatedly studied [15], [1], the goal of calibration is to identify error in a signal. Starting with some representation of the error, derived via physics or otherwise, the sensor is placed in a carefully controlled environment where precise and accurate measurements can be taken. Residuals are calculated from the difference between the true and the expected readings, and the error model is fit to the residuals. We depart from a calibration approach in two ways. First, the requirement of special equipment in calibration to obtain ground truth information is an overhead. We explore acquiring useful models in the absence of ground truth. Second, we make no distinction between signal and noise. Given identified errors, simulating is simple: clean data is generated to which noise is added, which is what is done in the simulators [10, 6]. Our approach is different in that we choose parameters to represent the entire signal directly. We use data-driven methods to learn how these parameters vary with state.

One way to characterize approaches to modeling for simulation is based on the assumed knowledge of state. Calibration, when valid over all of state space, is the most powerful. Within limits of applicability of the error models, results from calibration can be used to predict sensor output at any state. [2] solves the problem of lidar simulation in a different setting: training data is in the form of lidar scans taken outdoors, with no map, but accurate pose information is available via a high-quality INS. The focus is on modeling the environment, which is represented by a combination of fit planes and permeable ellipsoids. Realistic simulation in the vicinity of training poses is possible, but this solution does not generalize across environments. Our work, conducted and applicable indoors, considers another case: both the map and sensor pose are known nominally within reasonable error. We incur a loss in prediction power, but gain significant ease in data collection. This approach was motivated by the observation that while exact localization is difficult, current localization algorithms result in fairly good state estimates.

Also related is work on developing measurement models, which are an important part of probabilistic robotics. Models of high sophistication have been proposed, based on Gaussian processes [12, 9] and Bayesian networks [5]. By considering prediction of the measurement model covariance without ground truth, [14] addresses concerns similar to ours. In simulation our attention is both on modeling and validation. Validation of simulators is a topic that receives little attention. The experimental results in [6] are comparisons between simulators, not a comparison of simulation with reality. [7] introduces a synthetic benchmark for the evaluation of vision algorithm that use RGB-D data. Algorithms are compared on clean and noisy simulations, but there are no results showing how this correlates with real performance.

Part of the reason is the difficulty in recreating environments in simulation. In [2] extensive validation of lidar simulators is performed. Quantitative measures for comparing lidar readings are defined and computed for simulated versus real readings over a number of runs. Given the motivation of this work, we also explore comparing simulators at the level of algorithm behavior.

II. Formulation

We want to build a generative model of sensor readings given state, for simulation. Let state be denoted by $\mathbf{x}$. The state contains information about the sensor pose and the world in the form of a map. For example, in a 2D world, these would be (x, y, θ) and a set (m_x, m_y) of map coordinates respectively. Sensor readings are denoted by $\mathbf{z} = \{z^k\}$, where $k = 1 \dots K$ indexes bearings. The generative model is the distribution $p(\mathbf{z}|\mathbf{x})$. Data is assumed to be available in the form $\{\mathbf{x}_i, \mathbf{z}_{ij}\}$ where the $\{\mathbf{x}_i\} = 1 \dots N$ are different states and $\{\mathbf{z}_{ij}\}, j = 1 \dots M$ are multiple observations at the same state. In this work we require that the distribution over observations can be represented in a parametric form, $p(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{x}, \Theta)$, with Θ as the parameters.

We use a map that only stores occupied locations, and don't model factors like material reflectivity and illumination that affect observations. Recognizing the wide variability in the world, if a model must be built for specific conditions, we seek methods to make that process simple. An interesting question is: can a useful sensor model be constructed when there is noise in the state information? Suppose that true state in the data is unknown and only estimates $\{\hat{\mathbf{x}}_i\}$ are available. We may only have a nominal map and the approximate sensor pose. Under such uncertainty, an Expectation-Maximization approach comes to mind. For example, if the sensor were mounted on a robot and we had the control and observation history $\{\mathbf{u}_{1:T}, \mathbf{z}_{1:T}\}$, we could consider the Θ to be the maximizing parameters and states $\{\mathbf{x}_{1:T}\}$ the hidden variables. Given a sensor model, the E-step would involve state estimation. Given the distributions over state at each timestep, the M-step would be to maximize $\Theta = \arg\max'_\Theta P(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \Theta')$. However, we are interested in parameters that vary with state. In this work we do not assume a functional form for $\Theta(\mathbf{x})$ but use nonparametric regression. States are part of the data points that define the learned model, and can no longer be considered hidden variables.

The general problem of regression in the presence of measurement error has been studied in the literature on error in variables. Nonparametric prediction with input noise is in general a hard problem. Advanced methods [13, 3] allow input errors to be drawn from arbitrary distributions and construct unbiased estimators when multiple noisy observations of the same input are available. The resulting procedure is computationally intensive, however. A single prediction involves evaluating an integral as part of an inverse Fourier transform, a laborious task for a range simulator which needs to make hundreds of prediction at each state. Instead, we rely on results from regression calibration [4], a procedure in which the true state $\mathbf{x}$ is replaced by the noisy state $\hat{\mathbf{x}}$ and standard regression is performed. This simple procedure is most useful when errors are small, a characteristic we use to our advantage. Observing that localizing state within $1cm$ takes reasonable effort, but doing so within $1mm$ takes much more, this procedure of working with approximate state relieves some of the burden of model-building. It is important to keep in mind the limitations. Regression calibration requires that predictions are made on inputs observed with the same error as the training inputs. Strictly then, the range simulator does not estimate $p(\mathbf{z}|\mathbf{x})$, but $p(\mathbf{z}|\hat{\mathbf{x}})$. For a range sensor, approximate state knowledge during training also implies that bias in the readings and pose errors are interrelated. A bias predicted in range could be partly due to error in the training pose. Simulator results must be interpreted cautiously when testing applications for which highly accurate poses are critical, such as collision detection. With these caveats in mind, we claim that the formulation affords simplicity in model-building at the cost of a low margin of prediction error.

III. Implementation

We applied our framework to build a simulator for the rangefinder on a Neato robot, a planar sensor with 360 equispaced bearings. Details of the construction are described in [11]. A number of retrofitted Neatos are used as robots in a course on mobile robot programming at the authors' university. Students develop software for mobility and perception, and providing them with a simulator was one of the original motivations for this work. Despite its simplicity, there is sufficient richness in this task to illustrate the benefits of the approach we take. The Neato sensor's behavior is noisy enough in ways, such as large variances at distances greater than $4m$, that a naïve simulator does not capture.

Our formulation requires a parametric description of the observation distribution. We approximated each bearing's reading z^k as a random variable that takes a null value (in this case, 0) with some probability p_{null}, and with remaining probability is drawn from a normal distribution, $\mathcal{N}(\mu, \sigma)$. The resulting sensor model requires the prediction of three parameters for each bearing, $\Theta = (p^k_{\text{null}}, \mu^k, \sigma^k)$. As discussed earlier, we do not explicitly model error, as is traditional in calibration. The systematic bias in a bearing, for example, will be contained in the trend of the mean, μ. We then trained independent regressors for each parameter in the vector Θ. Across bearings, this assumes that the readings are independent given the state. Across a bearing's model parameters, this assumes that the process responsible for generating null readings is conditionally independent of that generating non-null readings, given the state. To make the task of learning simpler, the output variables were regressed on features computed from the state, $\mathbf{y} = \Phi(\mathbf{x})$. We found that the nominal range and angle of incidence of a ray, $y^k = (r^k, \alpha^k)$, were sufficient as features. For each bearing

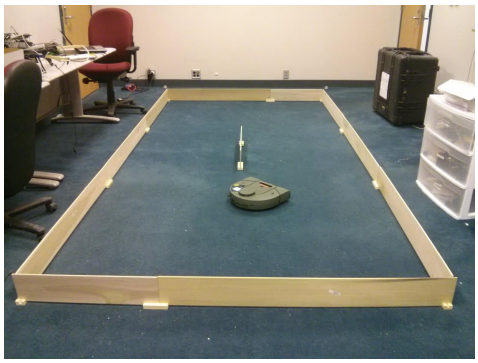


Fig. 1: A Neato in the data collection environment

they were computed from the approximate robot pose and the nominal map.

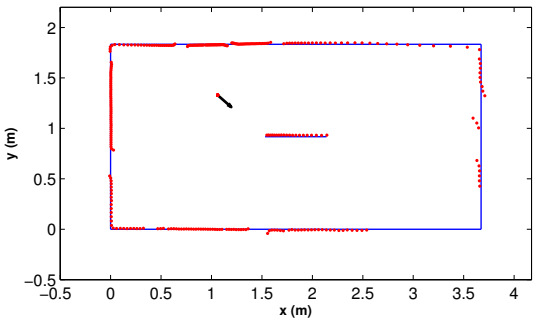
We collected data by recording ranges from a robot with a sensor placed in a custom environment, shown in Figure 1. It consisted of straight walls made from wooden boards placed to line up approximately with a nominal map, which was specified by a set of straight lines. From the dimensions of the boards, we estimated the error in map locations to be upper bounded by $1cm$. The robot was driven to different poses $\mathbf{x}_i$, and a number of scans $\mathbf{z}_{ij}$ logged. We localized the robot using odometry and range registration. To find an upper limit on the pose error, we placed the robot on a gridded floor where it could be repeatably positioned to within an accuracy of $1mm$, and compared this position to the localization estimate. The worst-case error in position and orientation of the localization estimate were $2.5cm$ and 2° respectively.

The environment was static during data logging, and we collected 100 training and 30 test instances. Each input-output relation was then learned using a locally weighted linear regressor. Hyperparameters such as the kernel width for predicting μ, σ and p_{null} were determined by 10-fold cross-validation over the training set. Final errors achieved over the parameters are in Table I.

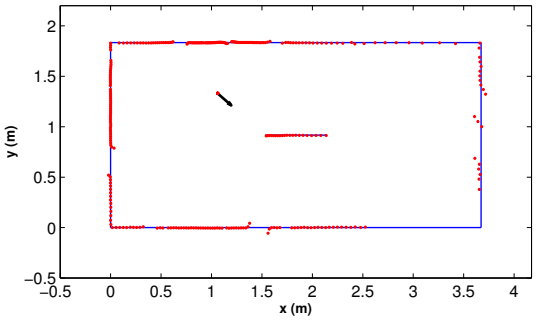
Parameter	μ	σ	p_{null}
Mean absolute error	$0.015m$	$0.001m$	0.0076

TABLE I: Test error on parameters

It is possible to work with a more sophisticated bearing model, set of features, or regressor. Our contribution is to recognize that model-building can proceed easily with approximate state information, not an expressive modeling technique that generalizes across sensors. Example test data and predictions are shown in Figure 2. The advantage to learning parameters for each bearing is that systematic biases can be reproduced. Certain bearings consistently record null readings, another behavior that is well-learned. The non-null readings seem geometric in nature, while the null readings are likely caused by physical obstructions to some bearings.



(a) Real ranges



(b) Simulated ranges

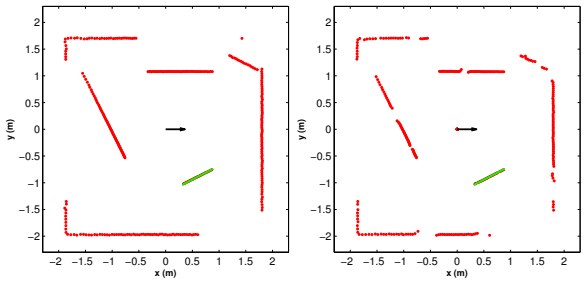
Fig. 2: Example test data and corresponding prediction. Bearing-specific bias and null readings can be simulated. The black arrow shows robot pose and the nominal map is the blue lines. Real ranges don't line up exactly with the map because the walls are not perfectly straight. To the simulator errors in state and bearing bias are interrelated.

IV. Experiments

In this section, we discuss studying simulator fidelity at the level of algorithm behavior. Broadly, we show that behavior in reality is approximated better on our learned simulator than on a baseline simulator. The baseline we implemented adds normal noise to ranges calculated by raycasting. The variance is state-dependent and based on expressions described in [8]. Two applications were examined. Each considered different algorithms and comparisons of fidelity. All software for communicating and working with the robots was written in MATLAB.

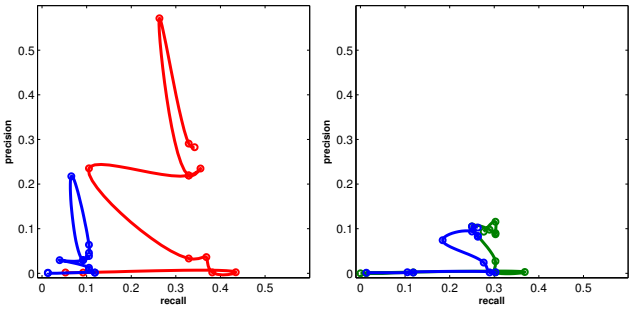
A. Detection

The first task considered falls into the class of detection algorithms and was representative of finding simple features in range scans. Specifically, we set up a task of searching for lines of a known length in the sensor's environment. These may stand for objects that need to be located accurately, such as targets for a robot forklift. We restricted the task to a simple world: the sensor is placed inside a square box, inside which exist lines of different lengths. Each length represents



(a) From baseline simulator (b) From learned simulator

Fig. 3: Sample scans in the detection dataset. The target is the green line. The black arrow shows the robot pose.



(a) Development on baseline simulator (b) Development on learned simulator

Fig. 4: Precision-recall trajectories. Performance on real data is in blue, the learned simulator in green and baseline in red. The upper limit on both axes is 0.6.

a 'class'. Multiple instances of one class are allowed. The goal is to detect all targets, which are lines of a particular length. The performance of a detection algorithm is summarized by the precision and recall, computed over a dataset. We generated different configurations of the world, and range scans from those configurations constituted our dataset. We first defined a distribution over the configurations, from which a large number of samples were drawn. From these, we picked 50 'diverse' configurations. These configurations were set up in simulation, and recreated in reality: the lines were wooden boards which we could accurately place on a gridded floor. The collected data resulted in three datasets: scans from reality, the learned simulator and the baseline simulator. Instances from the dataset are shown in Figure 3. We are particularly interested in how simulators compare to reality for naïve algorithms than high quality ones. We want the simulator to rapidly guide developers towards algorithms that will work well on the hardware with less (or no) changes. For this reason, we considered a simple line-finding algorithm that starts at a point and grows a line in either direction. Algorithm development is difficult to imitate. If every coded instance of an algorithm constitutes a set, a sequence of particular members of the set generated during development can trace a path in a space of performance metrics. Indeed, every individual development team would likely follow a different path in this space. We structured the problem by limiting the checks that could be activated and corresponding parameters that could be tuned. We omit details in the interest of space, but these included the maximum average distance of points to a candidate line, and minimum number of points in a candidate line. A 'start state' of the algorithm was chosen with all checks for false positives disabled, meant to mimic a first implementation. We then tuned parameters based on a fixed a strategy: first increase recall, then increase precision (at some cost to recall). At any stage, the performance of the algorithm is a point in the precision-recall curve. With continuing development, the algorithm can be imagined to trace out a 'trajectory' in the precision-recall space. This trajectory can be plotted for simulated and real datasets and is a visual aid. Figure 4a shows these trajectories for an algorithm developed on the baseline simulator. As is evident, performance on the baseline is overly optimistic. We repeated the exercise, by starting from scratch with the algorithm at the 'start state', and developing it on our learned simulator, trajectories of which are plotted in Figure 4b. Although it starts at the same point, the trajectory on real data (the blue curve) is different in either case, because algorithm development proceeds differently. For example, preprocessing the range data before detection was a crucial step for higher recall, a step suggested by the learned simulator but not by the baseline. Similarly, tolerance parameters for outlier rejection chosen on the baseline were too aggressive on the learned simulator. For detecting lines in range scans, we of course know how to construct a robust algorithm. The point of this exercise is that we can expect the learned simulator to be equally valuable on problems for which the developer is less skilled, or for which less is understood about the nature of a good solution.

B. Registration

The second task we considered belonged to the class of registration algorithms. Specifically, we looked at localizing with a range sensor as a robot moves along a trajectory. We approached this application differently compared to Section IV-A. While the point of the first experiment was to show the utility of the learned simulator by viewing algorithm development trajectories, in this experiment we compared the performance of one particular algorithm on simulated and real data. The algorithm, referred to as the registration-based filter, is a Kalman Filter. Its motion update leaves pose unchanged but adds noise from a large covariance. Its measurement update uses the output from an inner scan matching algorithm to fix the pose. We also conducted this experiment without ground truth, which complements our approach of training the simulator with approximate state information.

Suppose the robot were driven in a corridor, and the trajectory, denoted by a set of states for T timesteps,

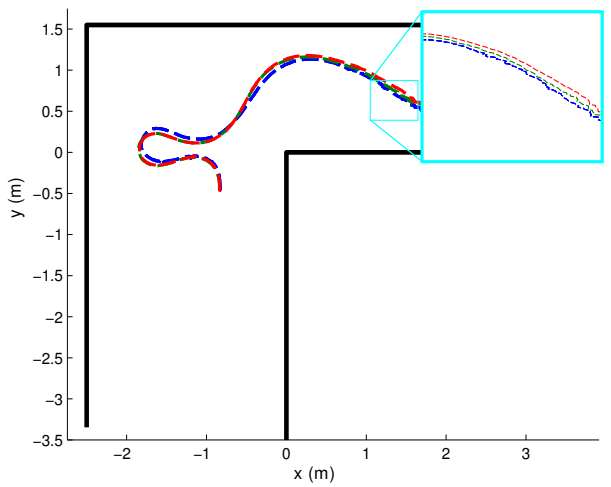
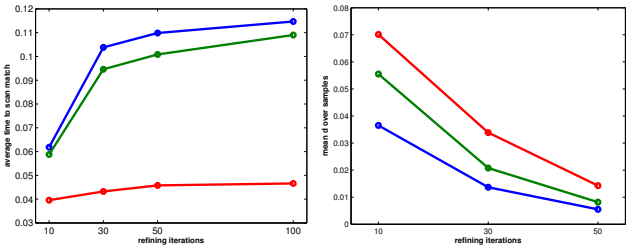


Fig. 5: Example trajectory estimates from the registration-based filter. Data from real scans is in blue, learned simulator scans in green and baseline in red. Detail of a section is also shown.

$\mathbf{x}_{1:T}$, was known. Corresponding to the trajectory, a set of real observations, denoted by $\mathbf{z}^{\text{real}}_{1:T}$, are generated. The registration-based filter can be run on these observations, resulting in a trajectory estimate, $(\rho^{\text{real}}_{1:T}, \Lambda^{\text{real}}_{1:T})$, where we use ρ and Λ to denote the mean and covariance from the registration-based filter respectively. How this estimate compares to the true trajectory would be a measure the real performance of the registration-based filter. We could also generate simulated scans, $\mathbf{z}^{\text{simulated}}_{1:T}$, at the states in the true trajectory. By running the registration-based filter on these simulated scans and computing $(\rho^{\text{simulated}}_{1:T}, \Lambda^{\text{simulated}}_{1:T})$, the performance in simulation could be obtained.

In the absence of ground truth, we calculated a distribution of real trajectories followed by the robot as follows: for a fixed set of controls (here, wheel velocities) $\mathbf{u}_{1:T}$, and initial conditions, running an Extended Kalman Filter (EKF) with only motion updates results in an estimated trajectory, $(\mu_{1:T}, \Sigma_{1:T})$. The noise parameters for the motion updates come from the robot's dynamics and were estimated empirically. μ_t and Σ_t are the mean and covariance of pose at time t respectively. Let this distribution of trajectories be D. Executing the controls on the real robot N times gives us a set of trajectories, $\{\mathbf{x}^{\text{real}}_{1:T,i}\}, i = 1 \dots N$, which are unobserved, and observed scans, $\{\mathbf{z}^{\text{real}}_{1:T,i}\}$. The result of running the registration-based filter on these scans is $\{(\rho^{\text{real}}_{1:T,i}, \Lambda^{\text{real}}_{1:T,i})\}$. We then used the EKF motion model to sample trajectories from the distribution, $\mathbf{x}_{1:T} \sim D$, and generated simulated scans, $\mathbf{z}^{\text{simulated}}_{1:T}$. Running the registration-based filter on simulated scans for N sampled trajectories results in the estimates $\{(\rho^{\text{learned}}_{1:T,i}, \Lambda^{\text{learned}}_{1:T,i})\}$ for the learned simulator, and $\{(\rho^{\text{baseline}}_{1:T,i}, \Lambda^{\text{baseline}}_{1:T,i})\}$ and the baseline simulator. We picked a natural corridor and a non-trivial trajectory for the robot, chose $N = 20$ and performed computations on recorded logs. Examples of estimates from running the registration-based filter are shown in Figure 5.



(a) Average time for scan matching algorithm to run (b) Average point-wise trajectory metric

Fig. 6: Measures versus the refining iterations of the scan matching algorithm. Results on real data are in blue, the learned simulator in green and baseline in red.

The important component of the registration-based filter, whose behavior is interesting to study, is the inner scan matching algorithm. The output from the scan matching algorithm is used in each measurement update of the registration-based filter. We implemented the scan matching algorithm as a gradient descent in the robot's pose space. Each instance of matching runs for a maximum number of iterations, called the refining iterations (the descent may converge in fewer iterations). There is a trade-off here: a large number of refining iterations results in a better pose estimate, but the robot will have moved more and the estimate gets delayed. We picked a set of refining iterations: $(10, 30, 50, 100)$, and calculated the average time for the scan matching algorithm to return a solution when producing the estimate $(\rho_{1:T,i}, \Lambda_{1:T,i})$. This was then averaged over the N samples. The results are plotted in Figure 6a. Time taken by scan matching remains low on the baseline simulator, but increases on the learned simulator and real data with increasing refining iterations. Encoder information arrives at $0.03s$ on the robot. If the baseline simulator were trusted and the refining iterations set to 100, scan matching would take $0.1s$ on average, and a number of encoder messages would queue up during the computation in a single thread.

We also compared the change in quality of estimate with change in the refining iterations. Since there is no ground truth information, we considered the estimate obtained with the refining iterations set to 100, say $(\rho_{1:T}, \Lambda_{1:T})$, as a reference. If $(\rho'_{1:T}, \Lambda'_{1:T})$ were the estimate with the refining iterations set to a lower value, we computed a point-wise average distance between mean positions,

$$d = \frac{1}{T} \sum_t \sqrt{(\rho_{x,t} - \rho'_{x,t})^2 + (\rho_{y,t} - \rho'_{y,t})^2}$$

which was then averaged across the N trajectories. We could compute the distance weighed by the covariance matrices, but we found that measurements from scan match in reality and simulation had low covariance and were essentially certain. Trends are shown in Figure 6b. Once again, performance on the learned simulator is a better approximation to real performance.

All runs of the robot were conducted at low velocity to avoid effects of motion blur. While not explicitly present in our current simulator, the effects of motion blur are simple to recreate given a bearing-level generative model of the sensor. We do not present visualizations here due to lack of space.

V. Conclusion

We are not the first to show that it is possible to improve significantly on the real-world fidelity of lidar simulation by taking a data-based approach. However, we have also shown that it can be done without investment in high quality ground truth infrastructure. Furthermore, we have shown that such simulation tools are likely to improve the efficiency of the software development process because they expose the flaws in naïve approaches earlier in the development cycle and in a manner that mimics what would eventually happen when testing on the hardware. This is significant in part because it is possible to slow down time in simulation, to insert breakpoints etc. in order to subject software to scrutiny that would be impossible in real time on the hardware. Our results suggest that we do not necessarily have to choose between realistic data and convenience in debugging. Data logs are another way to address this tradeoff but data logs do not generalize easily beyond the trajectory driven when the log was created. Our simulator produces high fidelity, data log quality results, on any trajectory.

A number of avenues for future work exist. On the theoretical side, a formal characterization of prediction errors based on bounded input errors will be useful. Validation in more complicated environments, where recreating the state of the world in simulation is difficult, is particularly challenging. More generally an optimal simulation testbed for software development might replicate the entire transfer function of the environment well, including the imperfections and challenges of mobility (wheel slip), communications (line of sight, packet collision), and conventional vision (CCD blooming). Dynamic scenes, perhaps including other robots and actors add even more complexity, and the modeling of outdoor scenes including weather and atmospheric obscurants will certainly be very difficult to accomplish. We are encouraged that these initial results have shown some evidence of utility of the approach while we acknowledge that a complete solution amounts to a grand challenge for the entire field of simulation and modeling, even if limited to its application to robotics.

References

[1] Dean Anderson, Herman Herman, and Alonzo Kelly. "Experimental characterization of commercial flash ladar devices". In: *International Conference of Sensing and Technology*. Vol. 2. 2005.

[2] Brett Browning et al. "3D Mapping for high-fidelity unmanned ground vehicle lidar simulation". In: *The International Journal of Robotics Research* 31.12 (2012), pp. 1349–1376.

[3] Raymond J Carroll, Aurore Delaigle, and Peter Hall. "Nonparametric prediction in measurement error models". In: *Journal of the American Statistical Association* 104.487 (2009).

[4] Raymond J Carroll et al. *Measurement error in nonlinear models: a modern perspective*. CRC press, 2012.

[5] Tinne De Laet, Joris De Schutter, and Herman Bruyninckx. "A Rigorously Bayesian Beam Model and an Adaptive Full Scan Model for Range Finders in Dynamic Environments." In: *J. Artif. Intell. Res.(JAIR)* 33 (2008), pp. 179–222.

[6] Michael Gschwandtner et al. "BlenSor: blender sensor simulation toolbox". In: *Advances in Visual Computing*. Springer, 2011, pp. 199–208.

[7] Ankur Handa et al. "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 1524–1531.

[8] Martial Hebert and Eric Krotkov. "3-D measurements from imaging laser radars: How good are they?" In: *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*. IEEE. 1991, pp. 359–364.

[9] Jonathan Ko and Dieter Fox. "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models". In: *Autonomous Robots* 27.1 (2009), pp. 75–90.

[10] Nathan Koenig and Andrew Howard. "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sendai, Japan, 2004, pp. 2149–2154.

[11] Kurt Konolige et al. "A low-cost laser distance sensor". In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE. 2008, pp. 3002–3008.

[12] Christian Plagemann et al. "Gaussian Beam Processes: A Nonparametric Bayesian Measurement Model for Range Finders." In: *Robotics: Science and Systems*. 2007.

[13] Susanne M Schennach. "Nonparametric regression in the presence of measurement error". In: *Econometric Theory* 20.06 (2004), pp. 1046–1093.

[14] William Vega-Brown and Nicholas Roy. "CELLO-EM: Adaptive sensor models without ground truth". In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE. 2013, pp. 1907–1914.

[15] Cang Ye and Johann Borenstein. "Characterization of a 2-D laser scanner for mobile robot obstacle negotiation". In: *ICRA*. 2002, pp. 2512–2518.

[16] Ji Zhang, Michael Kaess, and Sanjiv Singh. "Real-time Depth Enhanced Monocular Odometry". In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS 2014)* (2014).