2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Event-based 3D SLAM with a depth-augmented dynamic vision sensor

David Weikersdorfer[1], David B. Adrian[1], Daniel Cremers[2] and Jörg Conradt[1]

***Abstract*— We present the *D-eDVS*– a combined event-based 3D sensor – and a novel event-based full-3D simultaneous localization and mapping algorithm which works exclusively with the sparse stream of visual data provided by the *D-eDVS*. The *D-eDVS* is a combination of the established PrimeSense RGB-D sensor and a biologically inspired embedded dynamic vision sensor. Dynamic vision sensors only react to dynamic contrast changes and output data in form of a sparse stream of events which represent individual pixel locations. We demonstrate how an event-based dynamic vision sensor can be fused with a classic frame-based RGB-D sensor to produce a sparse stream of depth-augmented 3D points. The advantages of a sparse, event-based stream are a much smaller amount of generated data, thus more efficient resource usage, and a continuous representation of motion allowing lag-free tracking. Our event-based SLAM algorithm is highly efficient and runs 20 times faster than realtime, provides localization updates at several hundred Hertz, and produces excellent results. We compare our method against ground truth from an external tracking system and two state-of-the-art algorithms on a new dataset which we release in combination with this paper.**

I. INTRODUCTION

Simultaneous localization and mapping (SLAM) is one of the central tasks in robotics and computer vision which enables robots to explore and operate in unknown and unconstrained environments. While 2D or 2.5D SLAM which creates a bird-view map is an well-addressed problem [1], [2], [3], the full-3D SLAM problem has been tackled in recent years with the aid of combined color and depth (RGB-D) sensors like the PrimeSense device [4]. An important example for a 3D SLAM algorithm is KinectFusion [5], a dense 3D SLAM algorithm which uses iterative closest point to match depth images and a signed distance volume as a 3D map. The method of Bylow et al. [6] improves on Kinect-Fusion by using a more sophisticated representation of the signed distance function and a better optimization scheme. Kerl et al. [7] presented another dense visual SLAM method which uses the photometric and depth error to optimize the current position estimate. However many current dense 3D SLAM methods have the crucial disadvantage that they are very resource intensive and algorithms require dedicated GPU hardware which is expensive and has a high power consumption.

We use the low-cost embedded dynamic vision sensor (*eDVS*) [8] to intelligently reduce the amount of data which needs to be processed for tracking and mapping. Each pixel of the *eDVS* individually and asynchronously detects changes in the perceived illumination and fires pixel location events when the change exceeds a certain threshold. Thus events are mainly generated at salient image features like edges which are for example due to geometry or texture edges. In this sense we choose a middle ground between using only singular feature points and using all pixels. Depth information is an important requirement for 3D SLAM and as the *eDVS* is not a depth-sensor we will first augment pixel events with depth information by combining the *eDVS* with a separate, active depth-sensing camera like the PrimseSense sensor. This results in a sparse stream of 3D point events in camera coordinates which directly give the 3D position of salient edges in the 3D scene.

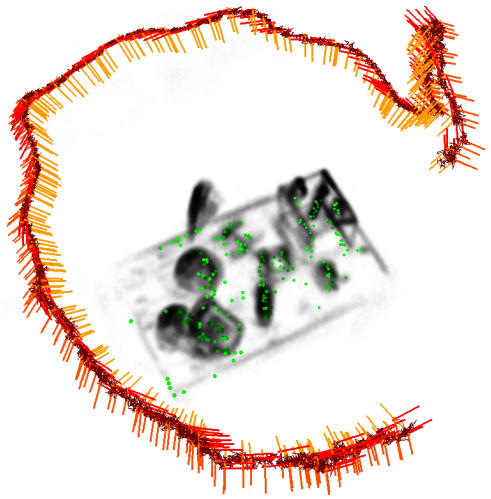

Fig. 1. Our event-based SLAM method *EB-SLAM-3D* creates a sparse 3D map (gray) which captures salient features like edges. The map is incrementally build using only 3D point events (green) provided by an dynamic vision sensor where each pixel event is augmented with depth information from an RGB-D camera. Due to properties of the dynamic vision sensor, *EB-SLAM-3D* creates excellent results for self-localization (red/orange).

The paradigm of event-based vision requires the development of new methods and we propose a novel event-based 3D SLAM algorithm (*EB-SLAM-3D*) which works solely on the sparse point event stream. Our algorithm uses a modified particle filter for tracking the current position and orientation of the camera while at the same time incrementally creating a map of the previously unknown environment. We update the internal belief state about location and map for every event and are thus able to provide pose estimates with very low latency. At the same time the computational task carried out for every event has minimal complexity allowing very

*This work was not supported by any organization

[1] Neuroscientific System Theory, Institute of Automatic Control Engineering, Technische Universität München, Munich, Germany

[2] Computer Vision Group, Computer Science Department, Technische Universität München, Munich, Germany

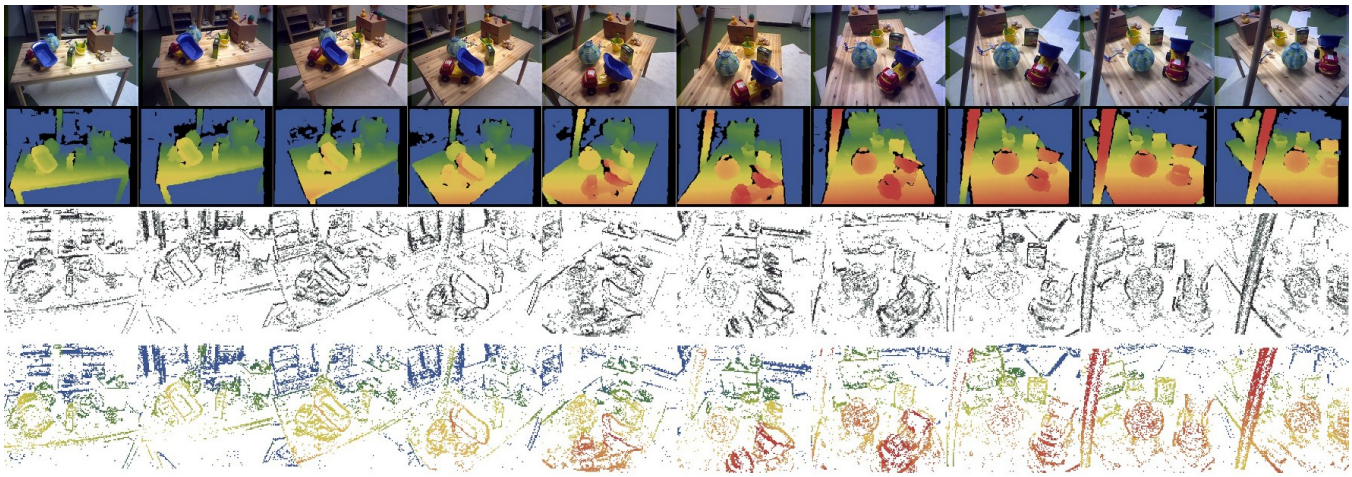

Fig. 2. Top: The PrimeSense color data stream for references. Middle top: The depth stream of the PrimeSense device with depth encoded as color (red to blue indicates near to far). Middle bottom: The event-stream provided by the *eDVS* sensor. Dark points indicate pixels which have fired an event. Bottom: The fused depth and event stream produces a sparse event-stream with depth information.

fast event processing. Low latency and low computational overhead are both important requirements for tight control loops found in quad-copters, or for autonomous vehicles which need to react quickly to changes in the environment.

This paper continues with a short introduction into event-based vision in section II and in section III the concept and calibration of the *D-eDVS* sensor is described. In section IV the event-based 3D SLAM algorithm is presented and tracking and map generation are explained in detail. The paper is concluded with a thorough evaluation of the *EB-SLAM-3D* algorithm in section V and a conclusion in section VI.

II. EVENT-BASED VISION

Conventional camera sensors provide recorded data as entire images at a fixed frequency, e.g. 30 or 60 Hz, regardless if the image has changed since the last frame. This results in a semi-continuous stream of highly redundant data which requires very high bandwidth. Furthermore they suffer from a fixed exposure time, resulting in a limited dynamic range and increasing the difficulty for image processing in unsteady light conditions.

Dynamic vision sensors [9] like the *eDVS* [8] are a novel type of gray-scale image sensors that completely abandons the concept of frame-based vision. All pixels of the *eDVS* operate asynchronously and individually integrate the measured light intensity over time. When the accumulated change for one pixel is higher than a threshold a pixel event is generated and inserted into the event stream. Such a pixel event consists of the pixel location $u \in \mathcal{R}$ on the sensor, in our case it has a resolution of 128×128 pixel, and a timestamp with micro-second accuracy indicating the time of occurrence of the event. Additionally a parity bit is reported which indicates if illumination has increased or decreased.

Typical image locations where events are generated are edges or other salient features with a high local contrast. Events only occur for moving entities in the scene or upon ego-motion of the sensor and for a static scene and motionless sensor no events, except noise events, are generated and no data will be transmitted. An important property of dynamic vision sensors is the fact that the number of generated events only depends on the moved distance and not on the velocity, but faster motions generate a higher number of events per seconds as the same distance is traversed in a smaller time. This counteracts problematic blurring in classic frame-based sensors, as all events necessary to understand a motion are detected and reported. This is a very beneficial property for example for accurate high-speed tracking as dynamic vision sensors are capable of producing up to one million events per second which is enough for very fast motions. A conventional frame-based sensor would need to operate with several hundred frames per second to produce similar results. As a summary, the *eDVS* directly creates a sparse stream of dynamic changes in hardware requiring no further software preprocessing to remove redundant information.

Fig. 2 shows an example of a stream of events generated by an *eDVS* in comparison to a classic frame-based camera like the PrimeSense RGB-D sensor. For visualization and printing several events are displayed together in one frame as if they have occurred at the same time, while in reality they occur one after the other and form a continuous stream of pixel events.

III. THE D-EDVS SENSOR

Raw events from an *eDVS* are only projected pixel locations which lack information about the exact world coordinate of the point which generated the event. The task of augmenting events with their respective depth is a non-trivial problem. One approach to acquire depth information is a fully event-based stereo vision set-up consisting of two *eDVS* sensors. While such a solution has been demonstrated [10], the capabilities are currently rather limited and not yet sufficient for deploying a 3D SLAM method. In this work we

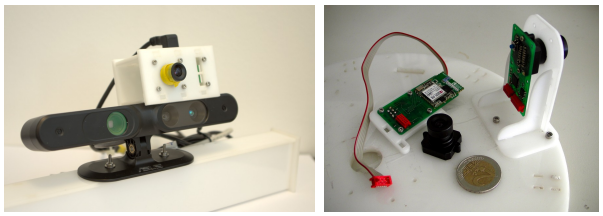

Fig. 3. The *D-eDVS* sensor (left) is a combination of an Asus Xtion RGB-D sensor (black casing) and a *eDVS* (white casing). The *eDVS* (right) is displayed with various accessories like WiFi module and lens mount, and a coin for size comparison.

chose to provide depth information by a dedicated depth sensor like the PrimeSense RGB-D sensor. Our implementation combines an *eDVS* with an Asus Xtion Pro Live (see fig. 3) running at a resolution of 320x240 pixels at 60 Hz. This implies a compromise between a frame-based and an event-based approach as depth information is not available between depth frames at the very moment events are generated. Depth values could be interpolated or extrapolated, but this would require additional computation power and special care to handle edges correctly. We chose a simpler approach by using the smallest depth value from the latest frame in a one-pixel neighborhood which works quite well for most cases. Thus events have high temporal resolution but low spatial resolution for motions parallel to the camera plane, and the opposite is true for motions orthogonal to the camera plane.

The main task of fusing the information of the *eDVS* and PrimeSense is the calibration of pixel correspondences, as we need to find the correct depth value for each event pixel location. In the following both cameras will be modeled as regular pinhole cameras $\mathrm{K}(f, c)$ where f is the focal length and $c \in \mathbb{R}^2$ the center of projection. The mapping of world points $x \in \mathbb{R}^3$ to image points $u \in \mathbb{R}^2$ is defined as $u = \mathrm{K}\,\mathrm{T}\,x$ where T is a transformation matrix representing a rotation and a translation. For real lenses it is necessary to also account for radial distortion. A simple distortion model like

$$L(u) = u\,(1 + \kappa_1 r + \kappa_2 r^2)\,, \quad r = \|u\| \tag{1}$$

with κ_1, κ_2 distortion parameters is sufficient for the *eDVS* and in case of the PrimeSense sensor the distortion is already compensated well enough in hardware.

Normally it is only possible to back-project image points to rays, but if depth information is available the inverse of the projection is well-defined. Thus in case of the depth sensor, we can compute $x = \mathrm{T}^{-1}\mathrm{K}_d^{-1}\,u_d$ of depth image points u_d to world points where K_d resp. K_e is the calibration matrix for the depth resp. event-based sensor. Subsequently we can compute corresponding *eDVS* image coordinates u_e as

$$u_e = L^{-1}(\mathrm{K}_e\,\mathrm{T}\,\mathrm{K}_d^{-1}\,u_d)\,. \tag{2}$$

This allows to establish a relation between the depth sensor and corresponding pixel locations on the event-based sensor.

To compute the internal camera parameters for K_e and K_d and the relative transformation matrix T, we record a set of corresponding pixel locations on both sensors and find the minimum of the least-square problem

$$\operatorname{argmin} \sum_{i=1}^{N} \left\| u_{e,i} - L^{-1}(\mathrm{K}_e\,\mathrm{T}\,\mathrm{K}_d^{-1}\,u_{k,i}) \right\|^2 \tag{3}$$

for the parameters f_e, c_e, κ_1, κ_2 and rotation and translation of T. The internal parameters of the PrimeSense device are known and do not need to be optimized. For our problem we solve eq. 3 by using local optimization as good initial values are available.

To find the necessary pixel correspondences we use a diode which emits pulses of light with a fixed and known frequency. This allows easy detection of the position of the diode in the event-based data stream by using frequency filtering. We reject all events at pixel locations where the time since the last event does not match the pulsing frequency (see fig. 4). On the other hand, the position of the diode needs to be detected in the depth image. Here we use built-in registration between color and depth images to be able to work with the color image instead of the depth image. We use the OpenCV library to detect a checkerboard in which the diode has been placed in the middle of the board (see fig. 4).

With the calibrated camera model it is possible to transform every pixel of the depth image to the corresponding pixel location in the event-based image. However to augment events with depth information the inverse operation is required which is a conceptually more difficult problem. We solve it by updating a 128×128 pixel depth-map for every new depth-frame which represents the current mapped depth values of the scene as seen by the *eDVS* sensor. For each new event we perform a look-up in the depth map and search a one pixel neighborhood for the smallest depth value which is is then used as event depth.

The bottom row in fig. 2 shows an example of a depth augmented event stream. This stream will be the only input used for the proposed event-based 3D SLAM – the dense depth or color image is not used any further.

IV. EVENT-BASED 3D SLAM

As the mathematical foundation for event-based 3D SLAM we use a dynamic Bayesian network and the Condensation particle filter algorithm [11]. In the dynamic Bayesian network used by the Condensation algorithm the current system state is modeled as a temporal random variable X_k

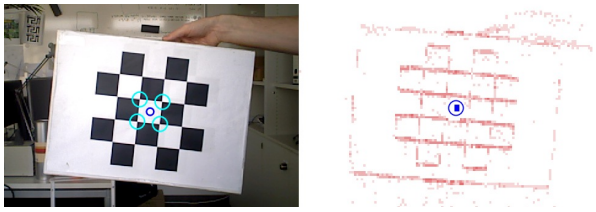

Fig. 4. To get corresponding points for camera calibration we track the position of a pulsed diode in both data streams by detecting a checkerboard in the color image (left) and by using a frequency filter on events (right). Only events which occur with the pulsing frequency of 100 Hz of our diode are used for the position estimate (blue) and other events (red) are rejected.

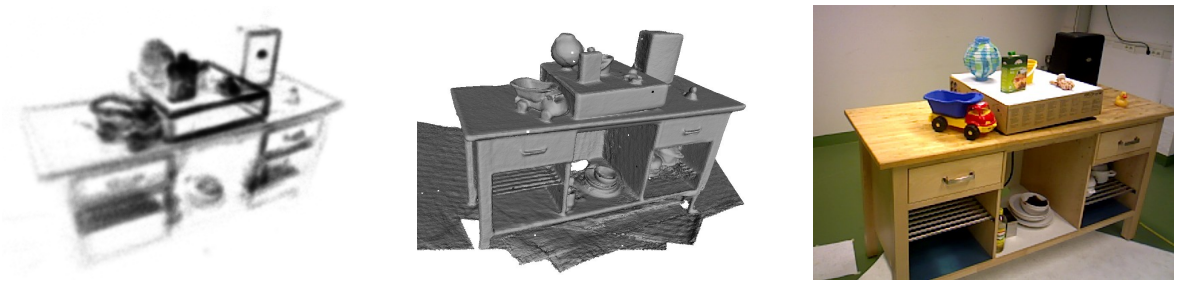

Fig. 5. Left: The event-based 3D map $\mathcal{M}$ used by our algorithm for tracking. Black color indicates a high probability for event-generation and thus a high score for particle evaluation. Middle: A mesh generated from the voxel-based map used by KinFu for comparison. Right: Color image for reference.

which is inferred using only the current measurement Z_k. The probabilistic density is realized by a set of N particles (p_i, s_i) where each represents a possible system state p_i together with a "score" $s_i \in \mathbb{R}_+$ indicating how well it represents the latest observed measurement. Particles for the next frame are selected based on this score, this step is called "resampling", and the procedure is iterated. In our case the system state is the special Euclidean group representing the current pose $p_i = (t_i, q_i) \in \mathrm{SE}(3)$ of the camera, i.e. its position $t_i \in \mathbb{R}^3$ and its orientation $q_i \in \mathrm{SO}(3)$. Normally the Markov assumption is used which states that only the last measurement Z_k has to be considered instead of the whole history of measurements $\mathcal{Z}_k = (Z_1, \ldots, Z_k)$. This leads to:

$$P(X_k|\mathcal{Z}_k) \propto P(Z_k|X_k) \int P(X_k|X_{t\ 1})\, P(X_{t\ 1}|\mathcal{Z}_{t\ 1})\, \mathrm{d}X_{t\ 1} \tag{4}$$

Here the sensor model $P(Z_k|X_k)$ defines the probability that a given state explains the current measurement. The motion model $P(X_k|X_{t-1})$ describes the dynamic change in the system and for this work no additional sensors are used, thus the motion model is simply a random diffusion process:

$$P(X_k \mid X_{t-1} = p_i) := \mathcal{N}(p_i, \Sigma) \tag{5}$$

The Markov assumption is reasonable for frame-based cameras as normally a complete image provides enough evidence to give good scores to particles. For the event-based case individual events are highly ambiguous and do not carry enough information to rate particles. Additionally the resampling operation is computationally expensive when executed several thousand times per second. For these reasons an incremental model is chosen where the Markov assumption is released and the score of particles does not only depend on the current measurement but also on the recent past of measurements [12], [13]. For each new event e_k particle scores are updated using an exponential decay model:

$$s_i = (1 - \alpha)\, s_i + \alpha\, P(Z_k = e_k|X_k = p_i) \tag{6}$$

where the decay constant α defines the influence of the current event compared to past events. An intuitive derivation of α is found with $\alpha = 1 - (1 - \beta)^{1/\kappa}$ which gives a decay constant where the last K events have an influence of $\beta \in [0, 1]$ percent on the total score s_i. This change with respect to a classical particle filter matches the event-based nature of the data stream and adds only minimal computational overhead.

As an additional measure to reduce the runtime of the algorithm the resampling step is not executed for every event but only after every K-th event. This is reasonable as individual events carry only little information and thus the probability density changes only little for only one event.

The map $\mathcal{M} : \mathbb{Z}^3 \to \mathbb{R}_+$ is modeled as a discretized probabilistic sparse voxel grid. It is updated alternately with the path, a concept known as Rao-Blackwellization [1], [2]. For event-based 3D SLAM each voxel in the map should indicate the probability with which this point would generate an event if the camera moves over it. This allows the formulation of the sensor model as

$$P(Z_k = e_k \mid X_k = p_i) \propto \mathcal{M}(\lfloor \tfrac{1}{\lambda}(t_i + q_i\, e_k) \rceil) \tag{7}$$

where $\lfloor x \rceil$ indicates the components of x rounded towards the nearest integer. The constant $\lambda \in \mathbb{R}_+$ is the size of voxel in world coordinates and a standard value is 0.01 m. The number of events generated at a specific spot depends on various factors like the amount of contrast towards the background for geometry edges, the relative change in intensity for planar image features or the orientation of a feature relative to the motion direction of the camera. However a model where each voxel counts the number of events observed at this location has proved to be sufficient. This gives a strikingly simple iterative map update rule for every new event:

$$\mathcal{M}(\lfloor \tfrac{1}{\lambda}\, p^*\, e_k \rceil) \mathrel{+}= 1 \tag{8}$$

where $p^* \in \mathrm{SO}(3)$ is the pose of the current best particle and $\lfloor \cdot \rceil$ indicates rounding to the nearest integer. Other modes to update the map are possible, like using the best particles weighted by their score or even using all particles. Fig. 5 shows an example of the event-based map generated by our algorithm. It can be seen how our map only captures salient features like geometry edges or texture features and ignores uniform solid surfaces compared to a solid mesh generated by other algorithms which represents all surface information.

A closer analysis of the runtime of our algorithm shows that a large share of the computation time is spent with particle diffusion in the motion model. Especially the rotation diffusion process requires the sampling of a uniformly

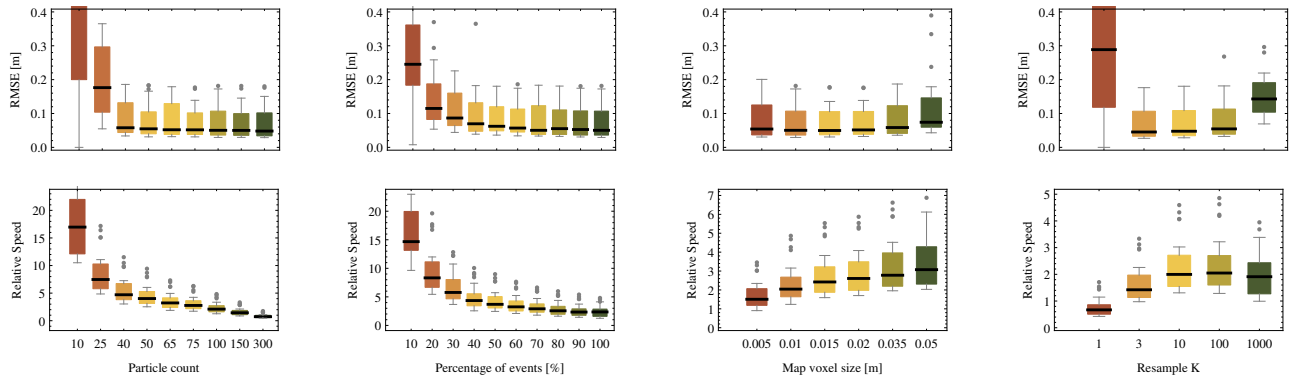

Fig. 6. Variation of parameters for our method: Number of particles (left), percentage of used events (middle left), voxel size (middle right) and number of events until resampling (right). The top row shows the RMSE and the lower row the runtime as a factor of realtime processing speed – a factor of 2 indicates our method runs two times faster than realtime. Boxes show the range of values for the mid two quartiles of takes in the dataset, the black bar shows the median value, whiskers have an interquartile range of 1 and individual dots show outliers.

distributed point on a sphere for the rotation axis and a normally distributed rotation angle. For higher performance we use the well known fact that two normal distributions can be added by adding their variances. Thus we collect events in mini-batches of B events, e.g. $B = 3$, and treat them as one package of information. This allows to execute normal diffusion only once per mini-batch with a standard deviation multiplied by $\sqrt{B}$. Additionally a modification of the score update rule from eq. 6 is required as the score update processes B events at once:

$$s_i = (1-\alpha)^B \, s_i + \frac{1-(1-\alpha)^B}{B} \sum_{j=0}^{B-1} P(e_{k+j} \,|\, p_i) \quad (9)$$

V. EVALUATION

For evaluation we compare our algorithm against ground truth and against two state-of-the-art algorithms: Kinect Fusion [5] and Kerl et al. [7]. As we require event-based data we recorded a new dataset consisting of five different scenarios and a total of 26 takes, where each take is one recording of length 20 to 40 seconds. Ground truth data for our dataset is provided by the marker-based motion tracking system OptiTrack V100:R2. Our dataset [1] is released to the public as part of this paper and consists of the event stream, ground truth data and full color and depth streams from the PrimeSense device for evaluation with other algorithms. For a comparison against KinectFusion we use the open source KinFu implementation of the Point Cloud Library (PCL) [14] and for Kerl et al. we use their publicly available open source implementation. A comparison of the absolute trajectory root-mean-square error (RMSE) of *EB-SLAM-3D*, KinectFusion and Kerl et al. compared to ground truth is shown in table I. We attribute a crucial contribution of our low RMSE to the event-based representation of motion which does not, like KinFu or Kerl et al., suffer from motion blur or too fast or non-constant velocities. In scenario 4 the sensor was moved around a large room which is a setting prone to loop-closure issues. The relatively high RMSE in scenario 4 shows that our algorithm can not fully handle this as no global path optimizer is used.

We report results for our algorithm with two parameter sets: A "quality" configuration which provides a low RMSE and is running faster than realtime, and a "speed" configuration which yields a slightly higher RMSE but runs more than 20 times faster than required for realtime processing. We evaluate runtime in form of a "realtime runtime factor" which is the quotient of the total time duration of the corresponding take and the time required for processing all occurred events. This is necessary as the event-rate is not constant and the computation time thus depends on the velocity of the sensor. All runtime durations for *EB-SLAM-3D* and Kerl et al. are measured on a single-core Intel i7 1.9 GHz CPU which consumes only 17 Watts of power and the GPU required by KinFu is a Nvidia GTX 670. Even in the "speed" setting our algorithm gives very good results which demonstrates the excellent performance of *EB-SLAM-3D*.

Additionally one can consider a comparison of power consumption: The graphics card used for KinFu requires 170 Watts of power and can barely process the data in realtime, while our algorithm runs 20 times faster than realtime with only 17 Watts of power, effectively consuming only about 1 Watt. In terms of memory requirements, the map has by far the biggest influence and requires around 12 to 30 MB of RAM depending on scenario and path. Thus, our sparse voxel grid requires significantly less memory than a dense voxelgrid which would require 64 MB for volume with side length 256 voxels and 512 MB for 512^3 voxels. Low memory requirements, high computational efficiency and low power consumption make *EB-SLAM-3D* an excellent candidate for autonomous robots or even embedded systems.

Additionally we analyze the influence of several parameters of *EB-SLAM-3D* on quality and runtime performance. Fig. 6 shows the RMSE and the realtime runtime factor for the following parameters: Number of particles N, percentage of events used, voxel size λ and number of events until

[1] http://ci.nst.ei.tum.de/EBSLAM3D/dataset/

TABLE I

EVALUATION OF *EB-SLAM-3D*, KINFU AND KERL ET AL.

Scenario	Takes	*EB-SLAM-3D* RAM	*EB-SLAM-3D* (quality) RMSE	Speed	*EB-SLAM-3D* (speed) RMSE	Speed	KinFu [14] RMSE	Failures	Kerl et al. [7] RMSE	Speed
1: "Table 1"	2	25 MB	3.1 cm	2.0 x	4.0 cm	20 x	N/A	2	7.7 cm	0.23 x
2: "Sideboard"	4	14 MB	4.0 cm	2.2 x	5.2 cm	23 x	4.3 cm	1	7.2 cm	0.23 x
3: "Table 2"	8	27 MB	4.9 cm	1.4 x	9.1 cm	16 x	16.5 cm	3	8.9 cm	0.28 x
4: "Room"	8	21 MB	13.4 cm	2.5 x	13.3 cm	27 x	28.8 cm	6	12.2 cm	0.23 x
5: "People"	4	15 MB	6.1 cm	2.2 x	7.0 cm	24 x	12.4 cm	3	4.5 cm	0.24 x

resampling K. The number of particles used in the particle filter have a direct linear influence on the runtime of the algorithm and our algorithm operates reliable even with low numbers of up to 40 particles. By varying the percentage of events a certain share of events is completely ignored and theses events are neither used for score update nor for map update. Reducing the number of used events makes the algorithm more unstable, but at the same time faster as no processing is required for ignored events. Evaluation shows that our algorithm works reliable with up to only 50% of events which inturn results in half the processing time. The voxel size defines the coarseness of our voxel grid and indicates the size length of a cubic voxel. The voxel size has mainly an influence on the memory requirements and reported memory values in table I are for $\lambda = 0.02$, the default value used in the "fast" parameter set. The last analyzed parameter in fig. 6 is K, the number of events until resampling. A value of 1 corresponds to a classical particle filter and evaluation shows that this is not a good choice for event-based sensors. This justifies the usage of the exponential decay model in eq. 6. Particle selection needs to consider at least several events for successful tracking and low runtime.

VI. CONCLUSIONS

The combination of dynamic vision sensors and RGB-D sensors is a promising opportunity for a new type of visual processing. The pre-processing provided by the hardware generates a continuous, sparse stream of 3D points events which captures only dynamic and salient information. The proposed event-based 3D SLAM algorithm is a strikingly simple algorithm which can produce excellent results twenty times faster than realtime without using special hardware like a GPU. Our algorithm is therefore especially suitable for small robots, flying robots and high-speed applications. As possible for future work we see the usage of a global optimizer to handle loop closure problems, and the augmentation of our map with sparse pieces of surface information to better predict possible collisions with scene geometry.

REFERENCES

[1] A. Doucet, J. de Freitas, K. Murphy, and S. Russel, "Rao-blackwellized partcile filtering for dynamic bayesian networks," in *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000, pp. 176–183.

[2] G. Grisetti, C. Stachniss, and B. Wolfgang, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics (T-RO)*, vol. 23, pp. 34–46, 2007.

[3] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *AAAI/IAAI*, 2002, pp. 593–598.

[4] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," *IEEE Conference on Computer Vision and Pattern Recognition*, 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5995316

[5] R. A. Newcombe, D. Molyneaux, D. Kim, A. J. Davison, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*. IEEE, 2011, pp. 127–136.

[6] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers, "Real-time camera tracking and 3d reconstruction using signed distance functions," in *Robotics: Science and Systems Conference (RSS)*, June 2013.

[7] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*, 2013.

[8] J. Conradt, R. Berner, M. Cook, and T. Delbruck, "An embedded aer dynamic vision sensor for low-latency pole balancing," in *IEEE Workshop on Embedded Computer Vision*, 2009.

[9] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x128 120db 15us latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid State Circuits*, vol. 43, no. 2, pp. 566–576, 2007.

[10] P. Rogister, R. Benosman, S.-H. Ieng, P. Lichtsteiner, and T. Delbruck, "Asynchronous event-based binocular stereo matching," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 23, no. 2, pp. 347–353, 2012.

[11] M. Isard and A. Blake, "Condensation conditional density propagation for visual tracking," *International journal of computer vision*, vol. 29, no. 1, pp. 5–28, 1998. [Online]. Available: http://scholar.google.de/scholar?hl=en\&q=Condensation+isard

[12] D. Weikersdorfer and J. Conradt, "Event-based particle filtering for robot self-localization," in *IEEE International Conference on Robotics and Biomimetics*, 2012, pp. 866 – 870.

[13] D. Weikersdorfer, R. Hoffmann, and J. Conradt, "Simultaneous localization and mapping for event-based vision systems," in *International Conference on Computer Vision Systems*, 2013.

[14] Kinectfusion implementation in the point cloud library (pcl). [Online]. Available: http://svn.pointclouds.org/pcl/trunk/.