Moving Volume KinectFusion

Henry Roth
roth@ccs.neu.edu
Marsette Vona
http://ccis.neu.edu/research/gpc

College of Computer and Information Science
Northeastern University
Boston, MA

Abstract

The recently reported KinectFusion algorithm uses the Kinect and GPU algorithms to simultaneously track the camera and build a dense scene reconstruction in real time. However, it is locked to a fixed volume in space and can not map surfaces that lie outside that volume. We present *moving volume KinectFusion* with additional algorithms to automatically translate and rotate the volume through space as the camera moves. This makes it feasible to use the algorithm for perception in mobile robotics and other free-roaming applications, simultaneously providing both visual odometry and a dense spatial map of the local environment. We present experimental results for several RGB-D SLAM benchmark datasets and also for novel datasets including a 25m outdoor hike.

1 Introduction

Many tools have been applied to the problem of accurate perception in 3D environments, including stereo cameras, laser range finders, depth cameras, and monocular cameras. One new tool that is now being explored is the Microsoft Kinect, a consumer depth camera made for gaming which can give as good or better data than more expensive solutions even as the sensor moves. While successive unaligned depth frames could be used directly for some perception tasks, in many uses it is necessary to spatially align and integrate the data as the camera moves. For example, a rough terrain walking robot needs to know about the ground under its feet, but the legs and feet themselves would obstruct downward facing cameras [10].

The alignment and integration problem has mainly been studied in the context of sparse SLAM-based methods [8]. Newcombe and Izadi et al's KinectFusion [11, 13] is an impressive new algorithm for real-time dense 3D mapping using the Kinect. It is geared towards games and augmented reality, but could also be of great use for robot perception. However, the algorithm is currently limited to a relatively small volume fixed in the world at start up (typically a $\sim$ 3m cube). This limits applications for perception.

Here we report *moving volume KinectFusion* with additional algorithms that allow the camera to roam freely (Section 2). We are interested in perception in rough terrain, but the system would also be useful in other applications including free-roaming games and awareness aids for hazardous environments or the visually impaired.

So far we have collected a total of 18 rocky terrain datasets comprising over 32k frames (18.1min) and an estimated 662m path length. (Though the Kinect cannot cope with direct sunlight it does work outdoors on a reasonably overcast day.) The richness of 3D depth features makes moving volume KinectFusion work very well on rocky terrain—no camera

http://dx.doi.org/10.5244/C.26.112

tracking failures were incurred, and reconstructed surfaces from the TSDF appear to be very high quality (quantitative analysis of the reconstructed geometry is still future work).

The processing requirements of our approach are similar to those of the original algorithm, which to the best of our knowledge can currently require a high-end desktop-class GPU to process data in real time. So while we can collect free-roaming datasets for later processing (Section 3), we have not yet achieved *mobile* KinectFusion.

We present experimental results (Section 4) for three datasets with ground truth from the TUM RGB-D SLAM dataset repository [18] and three new datasets we collected, including a 25m outdoor hike. We found the system to perform well as a combined visual odometry and dense local mapping algorithm. (Though it is possible to track the global camera pose, it drifts over time because no loops are explicitly closed.) We based our implementation on the open-source *Kinfu* code that has recently been added to the Point Cloud Library (PCL) from Willow Garage [15], and we have submitted our code for inclusion there as well.

1.1 Review of KinectFusion and Related Work

KinectFusion [11, 13] integrates depth maps from the Kinect into a truncated signed distance formula (TSDF) representation [3]. The TSDF is discretized into a voxel grid, typically $512 \times 512 \times 512$, that represents a physical volume of space (e.g. a 3m cube). Each voxel $\mathbf{v}$ contains two numbers: a signed distance d indicating how far that cell is from a surface and an integer weight w representing confidence in the accuracy of the distance. If $d < 0$ then $\mathbf{v}$ is "inside" a surface; if $d > 0$ then $\mathbf{v}$ is "outside" the surface. Only depth values within a truncation band $-T < d < T$ are stored (a typical value is $T = 0.03$m); the remaining voxels are sentinels with either $w = d = 0$ (uninitialized) or $d = T$ (empty space). The actual world surfaces are encoded as the zero crossings of the distance field and can be extracted by ray casting or marching cubes.

The computational expense of this approach is mitigated by a highly parallelized implementation on newly available GPUs with up to 512 or more floating point cores and several GB of memory. The original algorithm can typically process each new frame in well under the $\sim$ 30ms available before the next frame arrives. (Though the Kinect provides both RGB and depth images, only the depth data is used here.)

As the Kinect moves each new depth frame is used to incrementally localize its pose within previously observed geometry using the generalized iterative closest point algorithm (GICP) [17] with projective data association. The new readings are then integrated by sweeping through the TSDF: every cell which would appear in the camera is updated based on the previously stored values and the new depth map using a projective distance metric.

We recently learned that two other groups are also developing alternative approaches to translate the KinectFusion volume [9, 20]. A key distinction of our method is the ability to rotate the volume in addition to translation. Since the volume is rectilinear this can be useful to control its orientation, e.g. to maximize overlap of the camera frustum or to align the volume with task-relevant directions, such as the average ground surface normal in locomotion.

Beyond KinectFusion, Klein and Murray's parallel tracking and mapping (PTAM) system [12], and the more recent dense tracking and mapping (DTAM) reported by Newcombe et al [14], are both impressive. Instead of the Kinect they use a monocular camera, which has both advantages and disadvantages. Also highly related are visual and RGB-D (color+depth) SLAM [4] and visual odometry [10, 16] algorithms.

2 Moving Volume KinectFusion

KinectFusion works for fixed spaces, but in its original form it is not suitable for use with a widely moving sensor. The TSDF volume is mapped to an absolute volume in the world. Our approach allows the algorithm to handle a volume that moves arbitrarily on-line (Figure 1).

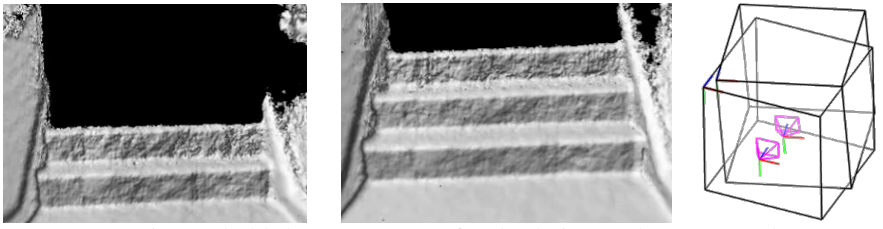

Figure 1: Remapping to hold the sensor pose fixed relative to the TSDF volume. Raycast images before and after a remapping show a third step coming into view as the volume moves forward. A reconstruction of the volume and camera poses shows that the volume-to-volume transform is calculated to maintain the camera at the rear center of the volume.

With our algorithm in place, the "absolute" camera pose C_g, a 4×4 rigid transform expressing the current camera pose in the very first volume frame, can be calculated at any time t as

$$C_g = P_0 \cdots P_{\mathrm{vf}(t)} C_t \tag{1}$$

where C_t is the current camera tracking transform from KinectFusion taking camera coordinate frame to its *parent volume frame*, each $P_{i>0}$ takes volume frame i to volume frame $i-1$, $P_0 = I_{3\times3}$, and $\mathrm{vf}(t)$ is a bookkeeping function that maps a depth image timestamp to the index of its parent volume. (Volume frames are generally sparser than camera frames.)

Moving volume KinectFusion both tracks *global* camera motion (equation 1) and simultaneously builds a spatial map of the *local* surroundings. However, this is not a true SLAM algorithm as it does not explicitly close large-scale loops and will inevitably incur drift over time. Rather, it can be considered a 6D *visual odometry* approach in that the camera pose ${}^aC_b$ at any time b relative to an earlier time a is

$${}^aC_b = C_a^{-1} P_{\mathrm{vf}(a+1)} \cdots P_{\mathrm{vf}(b)} C_b. \tag{2}$$

Of course the significant additional benefit beyond visual odometry alone is that a map of local environment surfaces is also always available in the current TSDF.

After the GICP tracking phase gives the current local camera pose C_t we determine if a new volume frame is needed by calculating linear and angular camera offsets l_d, a_d relative to a desired local camera pose C_s.

$$D = \begin{bmatrix} R_d & \mathbf{t}_d \\ 0 & 1 \end{bmatrix} = C_s^{-1} C_t, \quad C_t = \begin{bmatrix} R_t & \mathbf{t}_t \\ 0 & 1 \end{bmatrix}, \quad l_d = \|\mathbf{t}_d\|, \quad a_d = \|\mathrm{rodrigues}^{-1}(R_d)\| \tag{3}$$

A new volume frame is triggered if $l_d > l_{\max}$ or $a_d > a_{\max}$. We typically use

$$\begin{matrix} l_{\max} = 0.3\mathrm{m}, \\ a_{\max} = 0.05\mathrm{rad}, \end{matrix} \quad C_s = \begin{bmatrix} I_{3\times3} & \mathbf{t}_s \\ 0 & 1 \end{bmatrix}, \quad \mathbf{t}_s = \begin{bmatrix} W_m/2 \\ H_m/2 \\ -D_m/10 \end{bmatrix} \text{ for volume } W_m, H_m, D_m \text{ meters} \tag{4}$$

which is the default initial camera pose for Kinfu. This keeps the camera centered just behind the volume (Figure 1, right; note that the origin of each volume frame is the upper left corner of the volume with $\hat{\mathbf{x}}$ right, $\hat{\mathbf{y}}$ down, and $\hat{\mathbf{z}}$ pointing into the page). Other strategies

for determining C_s may make sense—for example keeping the camera centered in the volume, orienting the volume to task-relevant directions—but are subject to the practical (and application dependent) constraint that the camera must see scene surfaces within the volume.

To introduce a new volume frame we *remap* [2] the new volume from the old. We maintain a swap buffer in GPU memory the same size as the TSDF buffer for this; memory requirements for this large data structure are thus doubled but still feasible on current GPUs. After the remap the buffers are swapped and a new relative volume transform P_{n+1} is set as

$$P_{n+1} = C_t C_{t+1}^{-1} \tag{5}$$

where C_{t+1} is the new camera transform. Conceptually $C_{t+1} = C_s$, though we allow an offset in some cases as described in Section 2.1.

The core idea of remapping is to interpolate TSDF (d,w) values in the old volume at a grid of points corresponding to the samples (here the voxel centers) of the spatially rotated and translated new volume. (Points outside the old volume get the sentinel $(0,0)$.) Remapping—sometimes called *reslicing* for the 3D case—has been studied for medical images [7], but speed is often sacrificed for accuracy. Efforts have been made to improve the speed [5], but generally reslicing has not been done in real time. Here we require a fast parallel algorithm which is tuned for common-case TSDF data.

Our approach is hybridized in two ways (Algorithm 1). First, if $l_d > l_{\max}$ but $a_d \leq a_{\max}$ we use a fast and exact memory shift algorithm (Algorithm 2), otherwise we use a more traditional resampling (Algorithm 3). Second, during resampling we take advantage of the fact that in the common case much of the TSDF is either uninitialized or marked "empty": we do a nearest-neighbor lookup first, and only if that is within the truncation band do we continue with a more expensive (in this context) $2\times2\times2$ trilinear interpolation (Section 2.3).

Algorithm 1 Hybrid TSDF remapping.

function VOLUMEREMAP(rot $R \in SO(3)$, trans $\mathbf{t} \in \mathbb{R}^3$, orig vol $\mathbf{V}_a$, new vol $\mathbf{V}_b$)
  if $\|\text{rodrigues}^{-1}(R)\| > a_{\max}$ **then** VOLUMEINTERP$(R,\mathbf{t},\mathbf{V}_a,\mathbf{V}_b)$
  else if $\|\mathbf{t}\| > l_{\max}$ **then** VOLUMESHIFT$(\text{round}(\mathbf{t}),\mathbf{V}_a,\mathbf{V}_b)$

2.1 Volume Shift (Translation Only)

When the TSDF is translated by integer voxel units no interpolation is needed. Data that gets moved outside the volume is lost, but the remaining data is copied exactly, not approximated.

For each camera pose we calculate l_d and a_d as in (3) and if $l_d > l_{\max}$ but $a_d \leq a_{\max}$ we trigger a volume shift. The new camera pose and volume transform are

$$C_{t+1} = \begin{bmatrix} R_t & \mathbf{t}_t - \text{round}(\mathbf{t}_s) \\ 0 & 1 \end{bmatrix}, \quad P_{n+1} = \begin{bmatrix} I_{3\times3} & \text{round}(\mathbf{t}_s) \\ 0 & 1 \end{bmatrix} \tag{6}$$

We copy memory from $\mathbf{V}_a$ to each x,y plane of $\mathbf{V}_b$ in parallel. (This could be done in-place but we need the swap buffer anyway for full remaps.) We found that synchronizing all threads before completing each plane helps performance by optimizing memory cache hits since each plane is stored contiguously.

2.2 Volume Remap (Arbitrary Rotation and Translation)

When $a_d > a_{\max}$ a full interpolating remap is required. In this case

Algorithm 2 Voxel shift algorithm for translating a TSDF.

function VOLUMESHIFT(translation $\mathbf{t} \in \mathbb{Z}^3$, original volume $\mathbf{V}_a$, new volume $\mathbf{V}_b$)
  both volumes are $W_v \times H_v \times D_v$ voxels
  for $k = 0$ to $D_v - 1$ **do** ▷ synchronize before each plane to optimize cache hits
    for each $\mathbf{v} \in [0 \dots W_v) \times [0 \dots H_v) \times \{k\} \subset \mathbb{Z}^3$ in parallel **do**
      $\mathbf{v}' \leftarrow \mathbf{v} + \mathbf{t}$
      if $\mathbf{v}' \in [0 \dots W_v) \times [0 \dots H_v) \times [0 \dots D_v)$ **then** $\mathbf{V}_b(\mathbf{v}) \leftarrow \mathbf{V}_a(\mathbf{v}')$
      else $\mathbf{V}_b(\mathbf{v}) \leftarrow (0,0)$
  swap$(\mathbf{V}_b, \mathbf{V}_a)$ ▷ $O(1)$ pointer swap

$$C_{t+1} = C_s, \;\; P_{n+1} = C_t C_s^{-1}. \tag{7}$$

The interpolating remap is significantly slower than shift by memory copy, both due to the added math for interpolation and especially the greatly increased memory bandwidth as multiple voxels in a neighborhood need to be fetched for each sample point. Memory address locality is not as easily exploited due to the rotation. (It may be possible to use GPU texture fetch hardware, which has specialized caching for 2D arrays, but it is unclear to what extent this will also benefit 3D arrays.) Finally, as for any resampling, approximations are inherent.

Algorithm 3 Voxel interpolation for rotating and translating a TSDF.

function VOLUMEINTERP(rot $R \in SO(3)$, trans $\mathbf{t} \in \mathbb{R}^3$, orig vol $\mathbf{V}_a$, new vol $\mathbf{V}_b$)
  both volumes are $W_v \times H_v \times D_v$ voxels
  for each $(i, j) \in [0 \dots W_v) \times [0 \dots H_v) \subset \mathbb{Z}^2$ in parallel **do**
    for $k = 0$ to $D_v - 1$ **do**
      $\mathbf{v} \in \mathbb{Z}^3 \leftarrow (i, j, k)$, $\mathbf{w} \in \mathbb{R}^3 \leftarrow R\mathbf{v} + \mathbf{t}$, $\mathbf{v}' \in \mathbb{Z}^3 \leftarrow \lfloor \mathbf{w} \rfloor$
      if both $\mathbf{v}'$ and $(\mathbf{v}' + \mathbf{1})$ in $[0 \dots W_v) \times [0 \dots H_v) \times [0 \dots D_v)$ **then**
        $\mathbf{n} \in \mathbb{Z}^3 \leftarrow$ round$(\mathbf{w})$
        if $\mathbf{V}_a(\mathbf{n})$ is a sentinel **then** $\mathbf{V}_b(\mathbf{v}) \leftarrow \mathbf{V}_a(\mathbf{n})$ ▷ see text
        else $\mathbf{V}_b(\mathbf{v}) \leftarrow$ trilinearly interpolate $\mathbf{V}_a(\{\mathbf{v}' + \mathbf{o}\})$ for $\mathbf{o} \in [0,1]^3 \subset \mathbb{Z}^3$
      else $\mathbf{V}_b(\mathbf{v}) \leftarrow (0,0)$
  swap$(\mathbf{V}_b, \mathbf{V}_a)$ ▷ $O(1)$ pointer swap

2.3 Fast TSDF Interpolation

The distance, weight (d, w) pairs stored at each voxel in the TSDF fall into three categories:

$$\textbf{uninitialized}: d = 0, w = 0; \;\; \textbf{empty}: d = T, w > 0; \;\; \textbf{in-band}: \; -T < d < T, w > 0. \tag{8}$$

The first two are sentinels indicating respectively that nothing is known yet about the voxel or that the voxel was traversed only by camera rays beyond distance T from a surface, measured projectively along the ray. (Recall that T is the truncation band half-width.) There are fundamental discontinuities in the TSDF between voxels of each type.

Noting that the remapping inner loop is memory bandwidth bound, we first tried simple nearest-neighbor resampling. This was relatively fast but usually insufficient; while it does not "smear" the discontinuities, the resulting quantization quickly corrupts the TSDF truncation band enough to cause tracking failures and obvious geometric artifacts.

Trilinear interpolation requires 8 fetches for the 2×2×2 neighborhood of each sample point, and is thus more expensive, but it significantly reduces artifacts in the truncation band. It may smear discontinuities between voxels of different types, but we found that the algorithm seems robust to such corruption. Some geometric artifacts ("marshmallowing") can develop after numerous large rotations, but we have not observed tracking failures clearly due to this, and in practical use cases like walking or hiking the effect appears small.

Reasoning that in typical scenes large portions of the TSDF will be uninitialized or empty, we created a hybrid interpolant to first check if the nearest neighbor is a sentinel and early-out if so. This gave a 43% speedup across all datasets, though it sometimes introduces further artifacts which manifest as "holes" in recovered surfaces. We found them to be minimal in most cases, though noticeable in some datasets with fast camera motions.

3 Datasets

We present results on 6 datasets, three from the RGB-D SLAM repository [18] and three which we recorded ourselves. Figure 2 shows some highlights. Each dataset consists of a sequence of 640×480 depth frames from a Kinect at the full framerate of $\sim$ 30Hz.

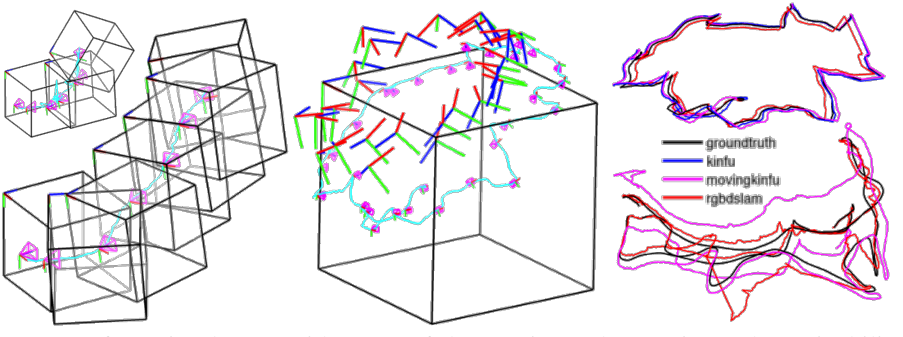

Figure 2: Left: stairs dataset with some of the moving volumes; inset shows inability of original Kinfu to track beyond the first volume (two later volumes mark camera tracking failures). Middle: freiburg2_desk dataset from the RGB-D SLAM repository [18]; here only the coordinate axes are drawn for volumes after the first. Right: spatial camera tracks aligned with the tool from [18] for freiburg2_desk (top) and freiburg1_room (bottom).

3.1 RGB-D SLAM Benchmarks

The RGB-D SLAM dataset repository [18] is an openly available collection of datasets prepared by the CVPR group at TUM for testing and comparison of SLAM algorithms based on combined color (RGB) and depth cameras like the Kinect. We selected three datasets which have corresponding mocap-based ground truth data:

freiburg1_xyz View of a desktop scene. (798 frames, 30s, 7.1m traveled)

freiburg2_desk Orbiting view of a desktop. (2964 frames, 99s, 18.9m traveled)

freiburg1_room Tour of a small room. (1360 frames, 48s, 16.0m traveled)

All three were captured indoors with a handheld Kinect. The original Kinfu implementation is appropriate for the first two, enabling direct comparisons with moving volume KinectFusion, but cannot handle the last due to extended camera motion.

3.2 Mobile Data Collection

The Kinect is normally constrained by a tether to wall power. Though some projects have used it on mobile robots [1, 6, 19], we are not aware of others who have adapted it for free-roaming handheld use. To this end we developed a simple system, shown in Figure 3, where the Kinect is powered by a lithium polymer battery, a tablet is attached to its back to provide a "heads-up" display, and a closed-lid laptop computer carried in a shoulder bag runs data collection software that stores all RGB and depth images to an SSD. This setup allows one person to conveniently hold and aim the Kinect, simultaneously control and monitor the data capture with the tablet, and walk freely without the constraint of any power cord.

Our system allows the collection of data in environments and scenarios that have usually been beyond the reach of the Kinect. As our group is especially interested in bipedal locomotion on rocky terrain, we are particularly interested to capture data outdoors. While it is generally understood that the Kinect does not work well in outdoor sunlight, we have found that it works fine on a moderately overcast day. Figure 3 shows an example of data collection outdoors in natural terrain, here a 25m hike up a rocky hill.

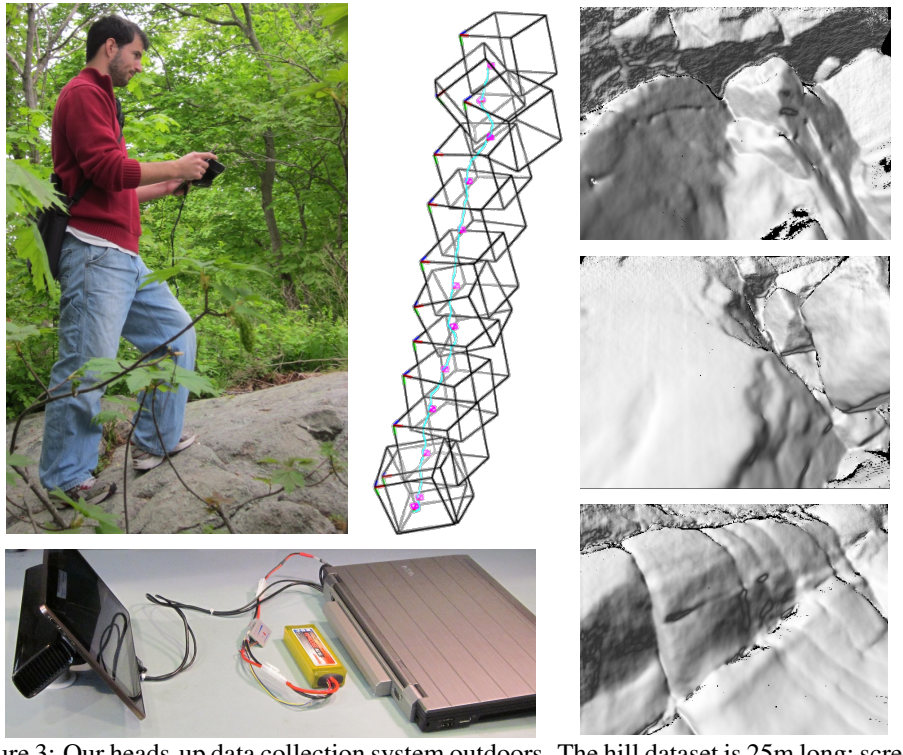

Figure 3: Our heads-up data collection system outdoors. The hill dataset is 25m long; screenshots are raycast terrain reconstructions from moving volume KinectFusion.

We collected three new datasets with this system; the original Kinfu implementation cannot handle any of these due to the free-roaming camera motion:

stairs A climb up two staircases in a hallway. (1313 frames, 44s, 11.3m est traveled)

hallway A stroll forward through a hallway. (1766 frames, 59s, 11.2m est traveled)

hill A hike up a rocky slope. (1109 frames, 37s, 25.1m est traveled)

The first two were captured indoors and the last outdoors. Path lengths are estimated from the moving volume KinectFusion reconstruction of the camera path.

4 Results

We measured the performance and tracking accuracy of our algorithm, comparing it with the original Kinfu implementation and with ground truth and reference results for RGB-D SLAM [4] where applicable. All results were measured using the typical values for $l_{\max}$, $a_{\max}$, and C_s given in Section 2. We have not yet quantitatively analyzed the geometric surface reconstructions; qualitatively they seem nearly as good as the original Kinfu output, with some degradations as noted in Section 2.3 above.

4.1 Performance

Table 1 gives processing times for each dataset with our algorithm enabled and disabled where possible. Tests were performed on an Intel Xeon W3520 processor (4 cores, 12GB RAM, 2.8GHz) and an NVidia GeForce GTX580 GPU (512 cores, 3GB RAM, 1.5GHz).

We also accounted the volume shifts and remaps. As expected the shifting is much faster (and also more predictable) at 6ms in every instance. The interpolating remap can take up to about 22ms, but averages about 15ms. Because the original Kinfu implementation can take up to 26ms per frame, true real-time performance is not attained as only $\sim$ 33ms are available between frames. However, only a small minority of frames require remaps so it is feasible to introduce a fixed-size frame queue (the alternative of dropping frames significantly reduced accuracy in our tests); the maximum observed latency was 14ms.

		ms/frame		latency	volume shifts			volume remaps		
Dataset	Kinfu	avg	max	max ms	num	avg ms	max ms	num	avg ms	max ms
stairs	orig[a]	21.1	26	–	–	–	–	–	–	–
stairs	mvol	25.7	46	12	9	6	6	81	16.2	21
fb1_xyz	orig	22.1	24	–	–	–	–	–	–	–
fb1_xyz	mvol	23.8	38	4	0	–	–	99	12.5	14
fb2_desk	orig	24.1	26	–	–	–	–	–	–	–
fb2_desk	mvol	25.1	46	12	2	6	6	228	16.7	21
hallway	mvol	25.6	47	14	6	6	6	166	16.2	22
hill	mvol	25.5	40	7	22	6	6	151	12.6	14
fb1_room	mvol	28.3	44	10	0	–	–	469	13.8	19

[a] Two camera tracking failures and subsequent resets.

Table 1: Runtimes for original KinectFusion and moving volume KinectFusion. The original algorithm was not measured on the latter three datasets as it incurred too many tracking failures due to a widely roaming camera. The fb (freiburg) datasets are from the RGB-D SLAM repository [18]. See text for machine configuration and runtime parameters.

4.2 Accuracy

We measured the tracking accuracy of moving volume KinectFusion on the three datasets for which ground truth was available, and we also compared with reported results for the RGB-D SLAM algorithm and with the original Kinfu implementation where possible. All of these results were enabled by the excellent data developed by the TUM CVPR group [18].

We report both absolute and relative errors. Absolute error is the distance between estimated and ground truth camera locations after rigidly aligning the trajectories. Relative error is the difference between (a) the relative pose ${}^{a}C_{b}$ (equation 2) for times a and $b = a + 1$s in the estimated trajectory and (b) the ground-truth relative pose from a to b. The former measures accuracy in a SLAM context where drift-free global alignment is expected; the latter does not penalize accumulation of drift and so is a better test for visual odometry alone.

The results indicate that the tracking accuracy of moving volume Kinect Fusion can compare favorably with the original algorithm; i.e. at least we have not obviously broken functionality where the original algorithm was sufficient to handle the camera motion.

For absolute error RGB-D SLAM outperforms moving volume Kinect Fusion, which is to be expected because we are not closing loops. Our approach fares better in the relative error measures, i.e. it performs reasonably well when considered a visual odometry algorithm.

		abs err [m]		rel err [m]		rel err [deg]	
Dataset	Algorithm	RMS	max	RMS	max	RMS	max
fb1_xyz	kinfu	0.021	0.070	0.029	0.069	1.7	4.4
fb1_xyz	mvkinfu	0.019	0.049	0.026	0.063	1.7	4.5
fb1_xyz	rgbdslam	0.013	0.035	0.021	0.048	0.9	2.3
fb2_desk	kinfu	0.095	0.254	0.026	0.127	1.2	6.2
fb2_desk	mvkinfu	0.117	0.331	0.020	0.070	0.8	2.6
fb2_desk	rgbdslam	0.095	0.146	0.017	0.092	0.7	3.0
fb1_room	mvkinfu	0.196	0.402	0.070	0.218	2.9	7.8
fb1_room	rgbdslam	0.101	0.437	0.095	0.558	3.2	14.0

Table 2: Tracking errors for datasets with ground truth from [13], analyzed with the tools from [13]. The original algorithm could not handle the last set. See text for definitions.

5 Future Work

We are aiming to use moving volume KinectFusion for mobile robot perception in natural terrain, including orienting the volume according to the terrain slope, gravity vector, and current heading. With a rectangular volume which allocates less voxels vertically and more horizontally this could increase the horizon available for locomotion planning. Another important next step will be the ability to use sensors other than Kinect in bright sunlight.

We also continue to optimize the code; for example it may be possible to use texture hardware on the GPU to accelerate neighborhood fetching and interpolation (though for this it may be necessary to alter the TSDF memory format). If sufficient speedups are found, and as GPU hardware—and our understanding of it—improves, it will also be interesting to experiment with higher-quality but more expensive interpolation schemes.

6 Conclusion

Moving volume KinectFusion frees the KinectFusion algorithm from one of the key constraints that made it unsuitable for use in mobile robotics and other free-roaming applications. By automatically translating and rotating the TSDF volume as needed we get both 6D visual odometry and an always-available local spatial map of the environment. Some tradeoffs are involved: up to about 14ms latency can be introduced and our interpolation scheme trades some quality for speed. The approach nevertheless seems feasible, with good results

on benchmarks with ground truth and on novel locomotion sequences including a 25m hike in the woods.

Acknowledgments

This research builds on the original KinectFusion [11, 13] algorithm and the Kinfu implementation by Anatoly Baksheev in PCL [15]. It was supported by NSF CAREER award 1149235 "Reliable Contact Under Uncertainty: Integrating 3D Perception and Compliance."

References

[1] Patrick Bouffard. Kinect + quadrotor. http://hybrid.eecs.berkeley.edu/~bouffard/kinect.html, 2010. Video demonstration of a battery-powered Kinect on a quadrotor.

[2] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning OpenCV, 1st edition*. O'Reilly Media, Inc., 2008.

[3] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 303–312, 1996.

[4] Nikolas Engelhard, Felix Endres, Juergen Hess, Juergen Sturm, and Wolfram Burgard. Real-time 3D visual SLAM with a hand-held RGB-D camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, 2011.

[5] J. Fischer and A. del Río. A fast method for applying rigid transformations to volume data. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, 2004.

[6] Willow Garage. Turtlebot. http://www.willowgarage.com/turtlebot.

[7] Joseph V. Hajnal, Nadeem Saeed, Elaine J. Soar, Angela Oatridge, Ian R. Young, and Graeme M. Bydder. A Registration and Interpolation Procedure for Subvoxel Matching of Serially Acquired MR Images. *Journal of Computer Assisted Tomography*, 19(2): 289–296, 1995.

[8] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *RGB-D Workshop at Robotics: Science and Systems (RSS)*, 2010.

[9] Francisco Heredia and Raphael Favier. Kinfu Large Scale in PCL. http://www.pointclouds.org/blog/srcs, 2012.

[10] Andrew Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3946–3952, 2008.

[11] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the ACM symposium on User interface software and technology (UIST)*, pages 559–568, 2011.

[12] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR '07, pages 1–10, 2007.

[13] Richard A. Newcombe, Andrew J. Davison, Shahram Izadi, Pushmeet Kohli, Otmar Hilliges, Jamie Shotton, David Molyneaux, Steve Hodges, David Kim, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011.

[14] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. DTAM: Dense tracking and mapping in real-time. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2320–2327, 2011.

[15] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point cloud library (PCL). In *IEEE International Conference on Robotics and Automation*, 2011. (http://www.pointclouds.org).

[16] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry part I: The first 30 years and fundamentals. *IEEE Robotics & Automation Magazine*, pages 80–92, December 2011.

[17] A. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Robotics: Science and Systems (RSS)*, 2009.

[18] J. Sturm, S. Magnenat, N. Engelhard, F. Pomerleau, F. Colas, W. Burgard, D. Cremers, and R. Siegwart. Towards a benchmark for RGB-D SLAM evaluation. In *RGB-D Workshop at Robotics: Science and Systems (RSS)*, 2011.

[19] Marsette Vona. The open hardware mobile manipulator (OHMM) project. http://www.ohmmbot.org, 2011.

[20] Thomas Whelan, John McDonald, Michael Kaess, Maurice Fallon, Hordur Johannsson, and John J. Leonard. Kintinuous: Spatially extended KinectFusion. In *RGB-D Workshop at Robotics: Science and Systems (RSS)*, 2012.