Large Scale Dense Visual Inertial SLAM

Lu Ma, Juan M. Falquez, Steve McGuire, Gabe Sibley

Abstract In this paper we present a novel large scale SLAM system that combines dense stereo vision with inertial tracking. The system divides space into a grid and efficiently allocates GPU memory only when there is surface information within a grid cell. A rolling grid approach allows the system to work for large scale outdoor SLAM. A dense visual inertial dense tracking pipeline incrementally localizes stereo cameras against the scene. The proposed system is tested with both a simulated data set and several real-life data in different lighting (illumination changes), motion (slow and fast), and weather (snow, sunny) conditions. Compared to structured light-RGBD systems the proposed system works indoors and outdoors and over large scales beyond single rooms or desktop scenes. Crucially, the system is able to leverage inertial measurements for robust tracking when visual measurements do not suffice. Results demonstrate effective operation with simulated and real data, and both indoors and outdoors under varying lighting conditions.

1 Introduction

(a) (b) (c)

Fig. 1: An example of the reconstruction result for an outdoor scene from 7000 stereo frames (approx 75 million vertices). a - b) Reconstruction detail of a scene with both shadow and harsh illumination, and snow on the ground. c) An overview of the camera path.

Lu Ma
University of Colorado - Boulder, Boulder, CO, U.S. e-mail: lu.ma@colorado.edu

Juan M. Falquez
University of Colorado - Boulder, Boulder, CO, U.S. e-mail: juan.falquez@colorado.edu

Steve McGuire
University of Colorado - Boulder, Boulder, CO, U.S. e-mail: stephen.mcguire@colorado.edu

Gabe Sibley
University of Colorado - Boulder, Boulder, CO, U.S. e-mail: gabe.sibley@colorado.edu

Large Scale SLAM is an important research area in robotics and computer vision. Perhaps the point based approaches [1–3] are the most popular ones for large scale SLAM. Normally, such approaches use a point cloud to reconstruct the scene and cannot reconstruct connected surfaces. These approaches register the point cloud in different views and present the reconstruction result as a point cloud. However, a connected surface is important for planning and control of robots.

Dense SLAM with volumetric representation have been popular in recent years [4–6]. Such techniques use a Truncated Signed Distance Function (TSDF) to represent the scene surface and incrementally refine it with the registered depth frames. Meanwhile, similar approaches have also been proposed in monocular SLAM [7, 8]. Usually, these approaches use a fixed amount of GPU memory for tracking and reconstruction; this hard constraint limits the size of the reconstructed scene and cannot be used for large scale dense SLAM.

Several systems have been proposed in order to reconstruct large scale scenes with volumetric approaches. [9, 10] proposed an octree based approach for indoor dense SLAM. [11–13] used a fixed bounded volume to represent portions of the scene and incrementally reconstruct it with a rolling scheme. However, these approaches mostly focus on the indoor scene and uses RGB-D sensors, which does not perform outdoor SLAM with stereo data. Meanwhile, these approaches heavily rely on ICP for tracking which are not suitable for outdoor environments due to the quality of the depth images from the stereo sensors. Besides, a combined ICP + RGB tracking approach [11] may also fail if the scene only contain simple geometric or color information.

Here we propose a new large scale dense visual inertial SLAM system that does not rely on active depth sensing. The system uses rolling grid fusion scheme which effectively manages GPU memory and is capable of reconstructing a fully dense scene online. The system obtains depth images from stereo matching [14] and simultaneously localizes the camera based on whole image alignment and inertial data while reconstructing the scene with SDF fusion. The system automatically saves and loads data from device, host memory and hard disk, and generates a mesh (.obj, .dae, .ply formats) of the large scene (e.g. 20 millions vertices) in seconds. Given these components, a wide range of applications can be developed, especially in robotics where the proposed system is capable of providing high fidelity meshes of any outdoor environment for use in path planning and control.

Perhaps the most similar system to ours is [12, 15–17]. There are, however, key methodological differences. 1) Our approach focuses on outdoor scenes and uses stereo data while [12, 15, 17] uses an RGB-D sensor and mainly focus on indoor scenes. 2) Our system uses an dense visual inertial stereo system for tracking while other rely solely on cameras, either ICP or RGB-D approach. 3) Our approach uses a simple rolling grid SDF pipeline for reconstruction while [15, 17] used a hashing scheme, [12] used a rolling SDF scheme and [16] uses a fix grid volume scheme.

The remainder of this paper is structured as follows: Section 2 briefly covers preliminaries of our approach. Section 3 covers the technical details of the Rolling Grid SDF approach. Section 4 covers the dense visual inertial tracking. Section 5 offers testing methods and discusses the system performance with indoor and

outdoor experiments. Section 6 addresses failure cases and limitations. Section 7 draws conclusions.

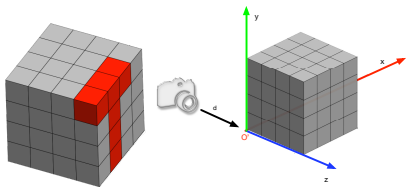
2 Overview

2.1 Grid Based Volumetric Representation

The proposed system uses a grid based volumetric representation, namely the *Grid SDF* $\mathscr{S}$ (see fig. 2), to reconstruct a 3D model of the scene in the current camera view. Each cell c in the Grid SDF $\mathscr{S}$ is a small *NxNxN* dimensional TSDF (Truncated Signed Distance Function) volume and contains a pointer to GPU memory. The *Grid SDF* $\mathscr{S}$ contains (x_g, y_g, z_g) cells in the each dimension. Assuming that the resolution of each voxel is r_v, the maximum size of the scene in each dimension is the number of cells in that dimension times the size of the SDF. For example, for the x dimension we have:

$$r_x = r_v * N * x_g. \tag{1}$$

The values of x_g and y_g are usually selected depending on the horizontal and vertical field of view of the camera, and z_g is based on the maximum depth measurement desired. This can be selected dependent on the maximum expected scene depth, or ideally, thresholded by the maximum depth uncertainty desired given the rig's stereo baseline. Notice that when initializing $\mathscr{S}$, the system does not allocate any GPU memory for cell c. Meanwhile, given the camera with an initial pose T_{wc}, the system defines a *Grid SDF* $\mathscr{S}$ as in Fig. 2, where the size of $\mathscr{S}$ is (r_x, r_y, r_z).



(a) (b)

Fig. 2: (a) An example of the *Grid SDF* $\mathscr{S}$. In this example, $\mathscr{S}$ has (e.g. $(8*8*8)$) cells in the x, y, z directions. The GPU memory of a cell g in $\mathscr{S}$ is not initialized (gray cells) until there is actual information available corresponding to c (red cells). (b) An example of the pose of $\mathscr{S}$ w.r.t the camera. The z axis of the initial pose of $\mathscr{S}$ starts from the minimum distance of camera range d_{min} to r_z

2.2 Grid Pose Representation

The system uses P_g to represent the global pose of the whole grid, $\mathscr{S}$, with $P_g = (0,0,0)$ being the world pose of the initial *Grid SDF*). P_l represents the local pose of a cell c within the grid. Thus, a cell c in the current camera view can be accessed by its local index and a voxel within the cell can also be accessed by P_g and P_l.

2.3 System Structure

The following flow chart (Fig. 3) shows the structure of the proposed system.

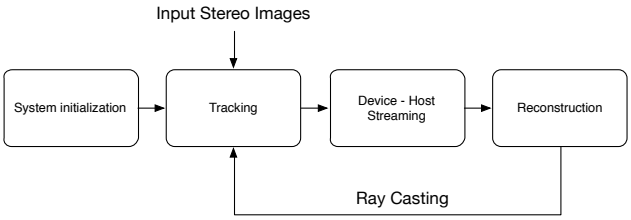


Fig. 3: Flow chart of the proposed system. After system initialization, the proposed system localizes the pose of cameras and incrementally reconstructs the scene with a rolling SDF scheme. Portions of the scene that are out of the camera view will be streamed from the GPU memory to the CPU memory (or the hard disk) directly. Such data can also be merged into a complete mesh via marching cubes.

Initialization The system first initializes a *Grid SDF* $\mathscr{S}$ without allocating any GPU memory for any cell c in $\mathscr{S}$.

Tracking Given the input stereo data, the system generates the depth images of the current frame via stereo matching and localizes the camera between the reference frame T_{wr} and the live frame T_{wl} via dense visual inertial tracking.

Rolling and Streaming Once the system updates the latest world pose of the camera, the system will check if rolling is needed based on the motion of the camera. If required, the system will stream the data of cells c that are out of the current camera view from the GPU memory to the CPU memory.

Reconstruction Once streaming is done, the system model can be updated via SDF fusion. Also, an updated view of the reconstructed scene is obtained via ray casting.

3 Grid Based SDF Fusion

3.1 Rolling Grid

In large scale outdoor SLAM, it is important to continuously perform mapping while at the same time reuse the GPU memory of voxels that have been taken out of the camera view. The proposed system achieves this via a rolling scheme which streams

the data of cells that are currently out of the camera view into the CPU memory and reuses the GPU memory of the cells.

To address this problem, we assume the initial pose of the *Grid SDF* $\mathscr{S}$ is the origin, and $\mathscr{S}$ moves with respect to the camera motion. The global pose of $\mathscr{S}$ in the x, y, z directions will increase by 1 if the camera moves $+r_x, +r_y, +r_z$ in the corresponding direction, and -1 if in the opposite direction.

Meanwhile, under the current camera view, the system can easily access a cell c of $\mathscr{S}$ via its local pose. However, based on the motion of the camera, a cell (e.g. $c^{'}$) in $\mathscr{S}$ may have moved out of the current camera view. To reuse the GPU memory of cell $c^{'}$ for a new cell $c^{''}$ in the current camera view, the system will stream the data of $c^{'}$ from the GPU memory to the CPU memory and reuse the same allocation for the new cell. In this case, we can no longer access $c^{''}$ via its local index directly in the current $\mathscr{S}$, implying that the real index of $c^{''}$ will be different from its local index. Figure 4 shows how the system computes the real index of a cell based on its local index during rolling.

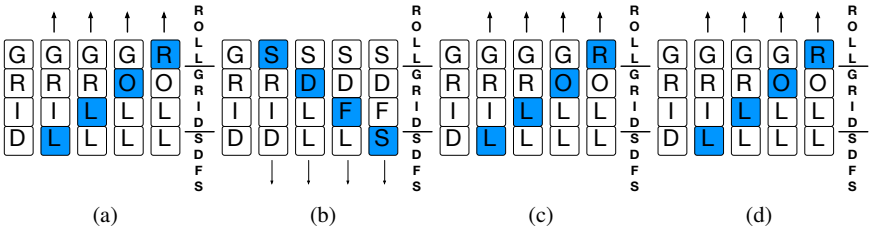


Fig. 4: An example of rolling *Grid SDF*. The camera is moving in the positive [(a) and (b)] and negative [(c) and (d)] directions; this example shows how the system reuses the GPU memory of the previous cells. Here we assume the number of cells is 4 and the initial scene that the camera sees is the letter sequence *GRID*. In each graph, the camera moves in the direction of the arrow. The white cells remain stationary within GPU memory. The blue cells store the scene that the camera sees in the current view, while the corresponding previous GPU-located cells have been streamed to the CPU memory. For example, in (a), column 2, the system sees *LGRI* in the current view, streams *L* from the GPU memory to the CPU memory, and then reuses the GPU memory location to store the new view *L* (in blue)

The proposed system performs rolling in a very straightforward way, as shown in Figure (4). Assume that the initial scene the camera sees is the word $'GRID'$. If the camera moves forward (e.g. Fig. 4 (a)), it will see the letters $'L'$, $'L'$,$'O'$,$'R'$ respectively. Here, each step (the minimum $r_x/x_v, r_y/y_v, r_z/z_v$) of the camera motion in a direction is considered a shift in that direction. Each time when the camera moves forward, the real index of the new cell (e.g. *L* in the second column of Fig. 4 (a)) will be saved to the cell which just moved out of view, and the corresponding previous cell (letter *D*) will be saved to the CPU memory. Now, the local index of *L* in the current *Grid SDF* should be 3, but its real index is 0 instead. The following

pseudocode shows how the system computes reused GPU memory by streaming cells that are out of the current camera view:

Algorithm 1 Compute the index of cells that needs to be streamed from GPU to CPU in a given direction (e.g. in the x axis)

Require: shift: s, previous shift: s_p, cell index: x, number of cells in one dimension x_g, stream flag: f
Ensure: s != 0 **and** $s < x_g$ **and** $s > -x_g$
  if $s > 0$ **then**
    if $s_p \geq 0$**and**$x \geq s_p$**and**$x < s_p + s$ **then**
      $f \leftarrow true$
    else if $x \geq x_g + s_p$**and**$x < x_g + s_p + s$ **then**
      $f \leftarrow true$
    else
      $f \leftarrow false$
    end if
  else
    if $s_p < 0$**and**$x \geq x_g + s_p + s$**and**$x < x_g + s_p$ **then**
      $f \leftarrow true$
    else if $x \geq s_p + s$**and**$x < s_p$ **then**
      $f \leftarrow true$
    else
      $f \leftarrow false$
    end if
  end if

Meanwhile, once rolling is performed, the real index of a cell can be computed directly by algorithm 2. Notice the voxel position is the real position of the voxel (3D point) in the space in the current *Grid SDF*.

Algorithm 2 Access a voxel in the *Grid SDF* by the voxel position (e.g. in the x axis)

Require: shift: s, local index: x_l, number of cells in one dimension x_v, real index: x_r
Ensure: $s < x_v$ **and** $s > -x_v$
  if $s > 0$ **then**
    if $x_l < x_v - 1 - s$ **then**
      $x_r \leftarrow x_l + s$
    else
      $x_r \leftarrow x_l - (x_v - s)$
    end if
  else
    if $x_l > -s$ **then**
      $x_r \leftarrow x_l + s$
    else
      $x_r \leftarrow x_l + x_v + s$
    end if
  end if

3.2 SDF Fusion

The system updates $\mathscr{S}$ by fusing every valid point from the stereo depth map I_d into $\mathscr{S}$ once T_{wc} is tracked:

$$\mathscr{S}' = \mathscr{F}(\mathscr{S}, I_d, T_{wc}) \tag{2}$$

Here, $\mathscr{F}(\cdot)$ is the SDF fusion operation. T_{wc} is the world pose of the camera in the live frame (i.e. current frame). The system also generates a virtual gray image I_v^g and depth image I_v^d by ray casting $\Upsilon(\cdot)$ [4]:

$$I_v = \Upsilon(\mathscr{S}, T_{wv}), I_v = I_v^g \cup I_v^d \tag{3}$$

where T_{wv} is the pose of the virtual camera.

Notice during fusion, the system will check every valid voxel position in the *Grid SDF* and project the voxel to 2D. If there is a valid 2D pixel in the current live image with a valid depth value, the voxel will be updated (a similar operation also happens during ray casting).

3.3 Device to Host Streaming

The proposed system automatically streams data from device memory to the host (CPU) memory if the data present in the *Grid SDF* is out of the current camera view. Once the memory block which hold the past SDF in the CPU memory is full, the system streams data of the cells which has the furthest distance to the current camera pose from the host memory to the hard disk. See Fig. 5.

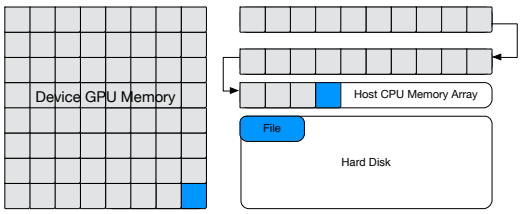


Fig. 5: Host - device streaming pipeline in the system. The blue block in the GPU memory will be streamed to the host CPU memory array when the data is out of the camera view and will be saved to the hard disk when the CPU memory array is full.

When the camera moves to a new location, the system checks if the data in the new location previously exists in the system. If it does, the system will reuse that memory and load it back from the CPU memory or the hard disk to the GPU memory. Reloading saved data helps to complete the model of the scene from different views. Notice that each time a cell file is saved in the host memory or the hard disk, the system indexes it with a global and local index which allows fast retrieval of

stored cells. Since all the SDF data is stored as individual cell files in the host memory or the hard disk, the system can merge any portion of the scene of interest into a mesh, which can be used later for any robotic application.

4 Dense Visual Inertial Tracking

Tracking is performed in a windowed dense visual inertial bundle adjuster. Visual-only frame-to-frame constraints are transformed into the IMU frame and added into the bundle adjuster as binary constraints. Inertial measurements between frames are integrated forming residuals against the estimated poses as seen in Fig. 6. Velocities and IMU biases are also estimated, and are carried through in the sliding window.

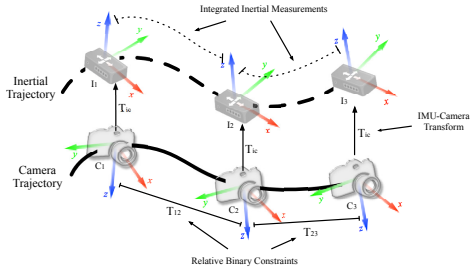


Fig. 6: Binary constraints from the visual tracker and integrated IMU poses, along with velocities and accelerometer+gyro biases are jointly optimized. The camera to IMU transform T_{ic} is calibrated offline.

Visual tracking is performed by a Lucas-Kanade [18] style whole-image alignment algorithm via the Efficient Second Order Minimization (ESM) technique [19], and a 6-DOF camera transform is estimated by minimizing the photometric error (e_v) between a reference image and the current live image:

$$e_v = \| I_{live} \left(\varphi \left(\hat{T_{lr}} \varphi^{-1} \left(\mathbf{u_r}, d \right) \right) \right) - I_{ref} \left(\mathbf{u_r} \right) \|^2. \tag{4}$$

The pixel $\mathbf{u_r}$ in the reference frame is back-projected φ^{-1} using the camera calibration parameters and the associated depth value d obtained by the stereo reconstruction algorithm. The 3D point is then transferred into the live frame via the estimated transform, $\hat{T_{lr}}$, and projected φ onto the camera.

The pose covariances from the visual tracking system are then added into the bundle adjuster, which runs once a sufficient number of frames and inertial measurements are obtained. The covariance of the inertial residual between two consecutive frames is dependent on the number of measurements between images, and as such must be carried forward during the integration process (Fig. 7). Details about inertial integration and error propagation can be found in [20].

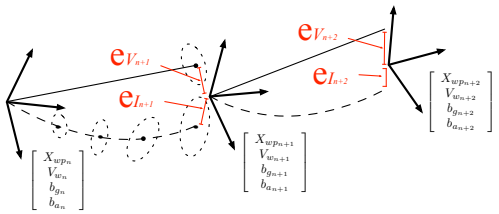


Fig. 7: Errors from the vision system (e_v) are formed by compounding the estimated relative transforms with world poses. Similarly, inertial errors (e_I) are formed by integrating inertial measurements. Uncertainties (shown as ellipsoids) are used to weigh in residuals for the estimation of the state parameters: world poses comprised of a translation (p) and rotation (q) vector ($X_{wp} = [p_{wp} \;\; q_{wp}]^T$), velocities ($V_w$), accelerometer biases (b_a) and gyroscope biases (b_g).

Inertial residuals between the parameters and the integrated state take the form of:

$$e_I = \left\| \begin{bmatrix} p_{wp} - \hat{p} \\ log\left(q_{wp}^{-1} \otimes \hat{q}\right) \\ v_w - \hat{v} \\ b_g - \hat{b_g} \\ b_a - \hat{b_a} \end{bmatrix} \right\|^2, \tag{5}$$

where $(p_{wp} - \hat{p})$ is the translation residual, $log\left(q_{wp}^{-1} \otimes \hat{q}\right) \in R^3$ calculates the rotation residual in $so(3)$, $(v_w - \hat{v})$ is the velocity residual, and $(b_g - \hat{b_g})$ and $(b_a - \hat{b_a})$ are the gyro and accelerometer bias residuals respectively.

A total of 15 parameters per frame are estimated during the sliding window optimization: 6 for pose parameters, 3 for velocities, 3 for accelerometer biases and 3 for gyroscope biases. Initial velocities as well as the biases are estimated and kept up to date as the sliding window shifts during execution. Given the size of the sliding window and the unambiguity of scale from the stereo vision system, no marginalization or conditioning is done on the sliding window as all parameters are observable.

The inclusion of inertial data enhances visual tracking in general, and in particular during fast camera movements and low textured areas. The addition of the IMU also speeds up visual tracking, since the typical coarse-to-fine pyramid scheme used in visual odometry is no longer required. Instead, the visual tracking is initialized with an estimated pose given by the integration of inertial measurements from the last frame up to the point where a new image is captured. In this way, only a refinement in the form of a few iterations at full image resolution is required for the final pose estimate.

5 Result and Discussions

The proposed system is tested by a hand held camera and a ClearPath Robotics Husky robot (Fig.8) with two Ximea (MQ013MG-ON) gray scale cameras and a Microstrain 3DM-GX3-35 Inertial Measurement Unit (IMU). The camera intrinsics as well as sensor extrinsics are calibrated offline with a method similar to [21], and the rigid sensor rig is attached to the robot via a T-mount.

(a) (b)

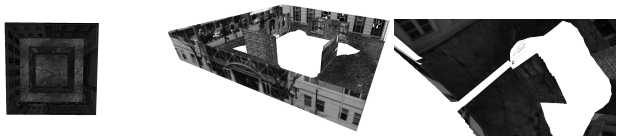
Fig. 8: An example of the system platform. An IMU and a calibrated stereo rig or an RGB-D camera is mounted on the robot which provides stereo and inertial data during navigation.

We implement the system using the GPU for the reconstruction pipeline and using the CPU with Intel Threaded Building Blocks for the visual inertial tracking pipeline. All the real-world datasets were captured using the stereo camera + IMU rig. The images were undistorted and scan-line rectified, and were later fed to a stereo matching technique [14] for depth map generation.

To evaluate the performance of the proposed system, we tested it with a simulated city-block dataset (15 meters by 15 meters in width and length, containing approximately 200 frames) with simulated IMU measurements and several real-world datasets (approx. 40 to 250 meters in length). For the real-world datasets, we captured a variety of indoor and outdoor scenes under different lighting and weather conditions (e.g. sunny and snow). To test the robustness of the proposed visual inertial tracking system, we especially test the system in a dark office scene (fig 10) and in an hallway with very simple geometry (fig 1), where either the traditional RGB-D approach or an ICP approach would easily fail. During the experiments, we set the maximum depth of voxels that fuse into the *Grid SDF* to 15 meters given the average maximum depth in all the scenes and in order to limit any potentially erroneous depth data from the stereo matching algorithm.

The dense visual inertial tracker initially performs visual odometry using a coarse to fine approach via an image pyramid. After a the minimum number of image frames is acquired, the sliding window kicks in and the image pyramid is no longer required since the IMU is capable of seeding the visual odometry optimization by providing a hint of the camera's pose. The window size used for all experiments was 15, with the minimum number of frames being 10.

We tested the accuracy of the proposed system with a simulated city block dataset. When compared against the ground truth depth map, the proposed system accurately tracks and reconstructs the city block scene with online performance. The path error is approx. 8cm after 60 meters of camera travel. When using the depth from the stereo algorithm, the path error is approx. 5cm after the same camera travel. Figure 9 shows the original mesh and the mesh generated by the proposed system. Notice in the detailed view the drift of the tracking system in the end of the reconstruction (fig 9(c)) showing the relative loop closure error.



(a) (b) (c)

Fig. 9: An example of the reconstruction result for the simulated city block data. (a) The original ground truth model. (b) An overview of the reconstruction result. (c) Close view of the reconstruction result showing loop closure error.

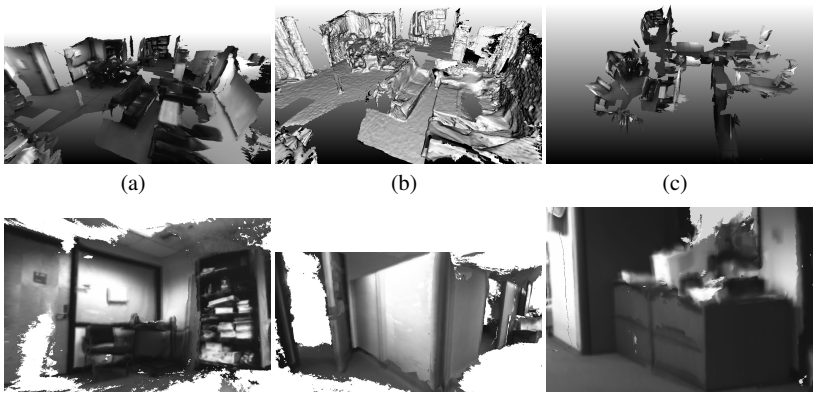
The proposed system also shows effective performance with real-world data. While the quality of the depth images generated from stereo matching is affected significantly by different lighting, texture and weather conditions, our system is capable of successfully reconstructing all large-scale outdoor scenes with high a quality mesh.

Figure 10 shows the reconstruction result of an indoor office scene (approx. 30m by 30m) from 6000 stereo frames. The system has a high precision which reconstructs fine details of objects in the scene.

While visual-only tracking may easily fail in real-world scenes with simple geometry, low texture or fast motion, the proposed visual inertial tracking shows a promising tracking result. Figure 1, 11 and 12 show the system successfully tracking under several difficult frames where the inertial measurements adjust the visual tracking result.

In general, the cell representation of the SDF volume massively saves GPU memory. When testing our simulated and real-world datasets, we set the resolution from 5mm to 25mm based on the dimension of the scene. In general, the proposed system requires around 650 to 1500 MB GPU memory to store voxels of the current camera view in a large scale scene while the regular SDF uses around 1000 to 3500 MB GPU memory, due to the fact that in general scenes, most of the voxels are empty.

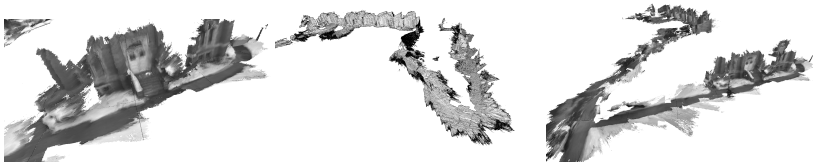
System Run-Time. We tested our system with a single NVidia TITAN GPU, Intel i7 quad-CPU desktop, using 640x480 pixel resolution input images and 2.5 cm resolution of voxels. Table 1 shows our system run-time in different stages. Except for final mesh generation, the system is capable of online performance.



(a) (b) (c)

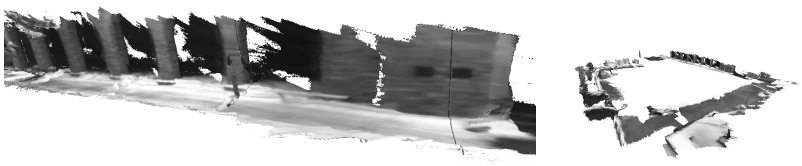
(d) (e) (f)

Fig. 10: An example of the reconstruction result for an office scene (approx. 5000 stereo frames (final mesh includes approx. 6 million vertices)) from a hand held camera (first row) and from the Husky robot (second row).



(a) (b) (c)

Fig. 11: An example of the reconstruction result for an outdoor snow scene from approximate 5000 stereo frames (final mesh includes approx. 32 million vertices). (a) A close look at a house in the scene. (b) An overview of the scene mesh. (c) An overview of the scene texture.



(a) (b)

Fig. 12: An example of the reconstruction result for an outdoor snow yard from approximate 7000 stereo frames (final mesh includes approxi. 15 million vertices). (a) A close look of the scene. (b) An overview of the scene mesh.

Table 1: System run-time

Stereo Matching, 1 frame	15 ms
Tracking (CPU), 1 frame	20 ms
Reconstruction, 1 frame	32 ms
Ray casting	8 ms
Device-Host Streaming 1 cell	0.01 ms
Generate cell to Mesh (e.g. 13 million vertices)	15 s

6 Failure Cases and Limitations

Although the system is robust to many real-world conditions, there are several limitations of our current work. The final reconstruction and tracking results depend heavily on the quality of the depth images which can be improved by [8]. The reconstruction can also be improved by adding loop closure by changing the local and global index of cells.

7 Conclusions

We present a large scale dense visual inertial SLAM system based on a rolling grid fusion scheme. As far as we know this is the first system to combine inertial tracking in a dense SLAM framework. The proposed system manages the space into small volume grids and only allocates GPU memory for cells if data exists. A large scale dense mapping solution is obtained via a rolling grid scheme with simple index computation while the device and the host memory automatically stream between each other in order to reuse the GPU memory. Depending on the requirements of an actual application, the system utilizes stereo cameras in both indoor and outdoor scenes. The system is tested in several outdoor and indoor scenes under different lighting (illumination changes), weather (e.g. snow, sunny), and motion conditions and shows promising results. In conclusion, the main contributions of the paper are: 1) A new large scale outdoor dense mapping system based on stereo data and 2) a new dense visual inertial dense tracking pipeline. We believe the proposed system is useful for outdoor scene reconstruction and especially for planning and control of high-speed ground vehicles.

References

1. A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6d slam3d mapping outdoor environments," *Journal of Field Robotics*, vol. 24, no. 8-9, pp. 699–722, 2007.
2. N. Fioraio and K. Konolige, "Realtime visual and point cloud slam," in *Proc. of the RGB-D workshop on advanced reasoning with depth cameras at robotics: Science and Systems Conf.(RSS)*, vol. 27, 2011.
3. H. Strasdat, A. J. Davison, J. Montiel, and K. Konolige, "Double window optimisation for constant time visual slam," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2352–2359.
4. R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and

tracking," in *Mixed and augmented reality (ISMAR), 10th IEEE Int. Symp. on*, 2011, pp. 127–136.

5. S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, *et al.*, "Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 559–568.
6. M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, "Real-time 3d reconstruction in dynamic scenes using point-based fusion," in *3D Vision-3DV 2013, 2013 International Conference on*. IEEE, 2013, pp. 1–8.
7. R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *Computer Vision (ICCV), IEEE Int. Conf. on*, 2011, pp. 2320–2327.
8. A. Concha, W. Hussain, L. Montano, and J. Civera, "Manhattan and piecewise-planar constraints for dense monocular mapping."
9. M. Zeng, F. Zhao, J. Zheng, and X. Liu, "Octree-based fusion for realtime 3d reconstruction," *Graphical Models*, vol. 75, no. 3, pp. 126–136, 2013.
10. F. Steinbrucker, J. Sturm, and D. Cremers, "Volumetric 3d mapping in real-time on a cpu," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2021–2028.
11. H. Roth and M. Vona, "Moving volume kinectfusion." in *BMVC*, 2012, pp. 1–11.
12. T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald, "Robust tracking for real-time dense rgb-d mapping with kintinuous," 2012.
13. R. Finman, T. Whelan, M. Kaess, and J. J. Leonard, "Efficient incremental map segmentation in dense rgb-d maps," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 5488–5494.
14. A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Computer Vision–ACCV 2010*. Springer, 2011, pp. 25–38.
15. M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 169, 2013.
16. S. Sengupta, E. Greveson, A. Shahrokni, and P. H. Torr, "Urban 3d semantic modelling using stereo vision," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 580–585.
17. V. A. Prisacariu, O. Kähler, M. M. Cheng, J. Valentin, P. H. Torr, I. D. Reid, and D. W. Murray, "A framework for the volumetric integration of depth images," *arXiv preprint arXiv:1410.0925*, 2014.
18. S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *International Journal of Computer Vision*, vol. 56, no. 3, pp. 221–255, 2004.
19. S. Klose, P. Heise, and A. Knoll, "Efficient Compositional Approaches for Real-Time Robust Direct Visual Odometry from RGB-D Data," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 2013.
20. N. Keivan and G. Sibley, "Asynchronous adaptive conditioning for visual-inertial slam." in *International Symposium on Experimental Robotics (ISER)*, 2014.
21. S. Lovegrove, A. Patron-Perez, and G. Sibley, "Spline fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras," in *Proceedings of the British machine vision conference*, 2013, pp. 93–1.