GPU-based real-time RGBD data filtering

Article *in* Journal of Real-Time Image Processing · September 2014
DOI: 10.1007/s11554-014-0453-7

Abdenour Amamra
Ecole Militaire Polytechnique
9 PUBLICATIONS **18** CITATIONS

J Real-Time Image Proc
DOI 10.1007/s11554-014-0453-7

ORIGINAL RESEARCH PAPER

GPU-based real-time RGBD data filtering

Abdenour Amamra · Nabil Aouf

Received: 23 January 2014 / Accepted: 9 September 2014

Abstract Commodity RGBD cameras such as Kinect sensor have recently proved a large success in many indoor robotics and computer vision applications. Nevertheless, professional applications cannot rely on their raw outputs because of the low accuracy. These consumer cameras can only produce precise depth measures within a small range. However, they do suffer from potential noise when the target is further away than permitted distance. The present paper proposes an innovative adaptation of Kalman filtering scheme to improve the precision of Kinect as a real-time RGBD capture device. We demonstrate Kalman filter adaptation to any Kinect-like camera and further justify the robustness of our approach with real experiments. A GPU implementation of the filter is also described with the different levels of optimisation.

Keywords RGBD · Real time · Tracking · 3D registration · Kalman filter · GPU

Abbreviations

GPU	Graphic processing unit
CPU	Central processing unit
RGBD	Red, green, blue and depth
RGB	Red, green and blue
TOF	Time of flight
FPS	Frames per second
IR	Infrared
3D	Spatial X, Y and Z three dimensions
KF	Kalman filter
MovAve	Moving average filter
RMSE	Root mean square error

A. Amamra (✉) · N. Aouf
Department of Informatics and Systems Engineering, Cranfield University, Shrivenham, Wilts SN6 8LA, UK
e-mail: a.amamra@cranfield.ac.uk

N. Aouf
e-mail: n.aouf@cranfield.ac.uk

1 Introduction

The concept of virtual reality is increasingly becoming a necessity for many civilian and defence applications, especially when importing concrete data of the scene into virtual models becomes possible [1]. Tools to capture 3D information of objects are required. Laser scanning technologies produce accurate scans [2], but they are expensive and require a level of expertise to manipulate them correctly [3]. Along with laser scanners, ultrasonic and radar scanners [4] are available for use as well. Existing multi-view methods [5] can produce acceptable results by reconstructing 3D models using two or more images taken for the same object from different viewpoints [6]. Nevertheless, image-based reconstruction process is computationally greedy, and therefore not convenient for real-time applications [7]. Furthermore, this technology presents some drawbacks such as sparse textures or complex occlusions among different views [6]. On the other hand, two other classes of range sensors, namely time of flight (ToF) and structured light cameras can be adopted as well [8]. The former captures reflected light and computes the distance between the sensor and the scene from the time that elapses between emission and reception [9]. The latter uses an IR projector that emits light patterns onto the scene [10]. These patterns are captured by an IR camera and then correlated to a reference stencil that has a known depth value. The deviation between snapshot range and the reference one produces a disparity image, which the sensor uses to deliver the actual range map of the scene [11]. In the present work, we aim to enhance the capabilities of commodity range cameras to derive accurate and trustworthy 3D scans. The device we used is Microsoft Kinect,[1] which is an off-the-shelf RGBD sensor that was first designed for interactive gaming. More importantly, it has proved a vast success among computer vision and graphics professional and research applications alike since it was first introduced. Moreover, accuracy itself is not sufficient for good 3D data capture; real-time performance should also be ensured in the architecture we design. In other words, the filtering algorithm should run at the same rate of capture without introducing latency to the whole system and by benefitting as much as possible from the maximum frame rate of the sensor. To this end, we propose a parallel design implemented on the GPU to clean the raw depth measurements output by Kinect. At this level, it becomes possible to embed the GPU-based filtering setup in dedicated cards for a self-contained capture/filtering system. This last configuration could be added to every RGBD setup as a pre-processing stage to feed the subsequent levels with neat point clouds.

The present work is divided into these sections. In the first one, “related work”, we discuss state-of-the-art research about RGBD data filtering. The architecture of our filtering scheme is fully presented in “System architecture”. We highlight “Kinect sensor technical specifications” as an example of RGBD camera. We present “Kalman filter” as a real-time refinement approach. Kalman filter is then leveraged to enhance the robustness of RGBD sensors in “Kalman filter on Kinect’s data”. “Kalman filter effect on RGBD data for moving vehicles tracking” illustrates how our findings about Kinect could be beneficial to recognise Kinect as an accurate real-time measurement device.

In “Kalman filter effect on RGBD data for depth images registration” we assess the visual quality of the geometry built upon filtered 3D data. In the next section “Results and discussion”, the precision of our solution is evaluated. Finally, in “Conclusion and future work” we highlight the benefits of our approach and present the future works we are targeting.

2 Related work

By nature, depth data are rough and noisy Fig. 1a. RGBD sensors in general are sensitive to noise because of their active nature. Filtering approaches aim to remove noisy data (outliers), clean the useful zones (inliers) and preserve the edges. In the presence of specular reflective or light-emitting objects, holes appear in the captured depth data. Hole filling is a challenging task because of the missing values. The recovery of the lost parts is generally based on some assumptions about the regions surrounding them and where the disparity data are available.

Despite the quantisation noise (noise created when reproducing the continuous real world with a discrete digital representation), recent research papers mostly use raw Kinect data without any filtering. However, limitations in the covered space and the maximum reachable depth need to be addressed. Other papers focus on Kinect depth data de-noising or propose data pre-processing stages as a basis for their applications. Menna et al. [12] present a detailed study on the precision of the Kinect depth map. However, no specific approach to depth map accuracy improvement has been proposed. They used a filtering approach based on spatio-temporal median computed from the motion vector. Camplani et al. [13] used an adaptive joint-bilateral filter that combined depth and colour by analysing an edge-uncertainty map and the detected foreground regions to improve the quality of the Kinect depth map.

Kalman filter uses a sequence of noisy measurements observed over time and produces statistically optimal estimates [14]. This filter is well known amongst navigation, guidance, communication and control researchers, since it helps enormously to predict and correct the noisy measurements. Kalman de-noising method has been recently implemented to clean Kinect data in a few works. Ling et al. [15] applied extended Kalman filter in a real-time 3D mapping framework on Kinect RGBD data. The authors proposed repetitive linearisation of the nonlinear measurement model to provide a running estimate of camera motion. Similarly, Thibault et al. [16] applied nonlinear Kalman filtering to generate accurate 3D maps. Furthermore, Sangheon et al. [17] proposed a 3D hand tracking method based on a Kinect and a Kalman filtering strategy. In all the cited works, Kalman filter was customised to fit the target application (3D mapping or tracking). However, the novelty of the present work originates from the fact that we uncovered some interesting characteristic properties of Kinect sensor and the behaviour of its output over time. We have also demonstrated that the depth measurements can be optimally filtered to feed several applications without any supplementary tuning of the parameters. Our modelling is useful for all the users of Kinect camera, in particular, as well as RGBD sensors, in general.

3 System architecture

The Kinect outputs three different streams of data[2] as illustrated in Fig. 2. The system we are aiming to design in the present paper targets only depth stream. Kinect sensor

[1] http://www.xbox.com/en-GB/Kinect, 2013.

[2] http://msdn.microsoft.com/en-us/library/jj131023.aspx, 2013.

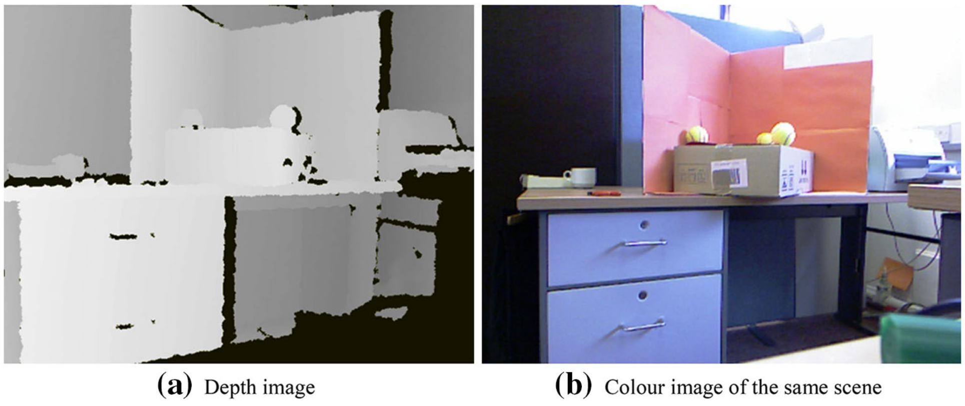


Fig. 1 Kinect depth and colour data

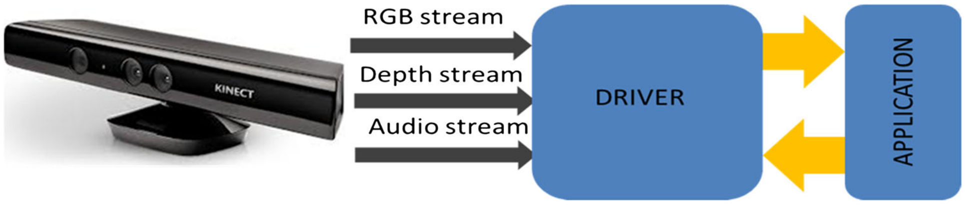


Fig. 2 Kinect data streams

has the advantage of working in real time at a frequency of 30 fps. Whenever a further processing is added to the line, a drop in frame rate occurs. Hence, to conserve the real-time nature of the solution, we must find a suitable hardware and software combination that best answers this need. From a hardware point of view, the same processing (kernel) could normally be applied in parallel on all the pixels of the depth image. In the implemented solution, we achieved a frame rate ranging between 25 and 30 fps, which is very convenient for the application level, as the filter computational load is relatively low during the whole capture. To reach real-time performance when processing huge amounts of 3D data, we need enough computation power to handle the flow of depth images. In the literature, GPUs have provided many advantages, while CPU has proved to be too slow to handle considerable amounts of data. As a consequence, we designed a series of algorithms to be embedded in the graphic processor. This allows us to fully benefit from the maximum frame rate of the camera (30 fps) which would be otherwise impossible to reach with classic CPU implementations.

Figure 3 shows the stream of depth data from the camera, to the parallel filtering stage, and then finally to the application level. This architecture could be easily integrated as an independent data enhancement stage into the driver of any Kinect-like camera. The core algorithm needs only to be initialised with the appropriate camera calibration parameters specific to the sensor.

From the software side, we adapted a recursive state-transition filter to Kinect's depth data. However, this was not a straightforward task, since some complex conditions need to be fulfilled. In addition, this kind of adaptation has not been previously discussed in the literature. Thus, we developed our own mathematical formulation to adequately fit the problem into the filter's model.

4 Technical specifications of the Kinect sensor

The Kinect sensor is an RGBD camera which has the ability to capture both a depth map of the scene and an RGB colour image at a frame rate of 30 fps. The sensor includes the following:

- An IR projector: projects IR patterns on the scene.
- An IR camera: captures the reflected light of the projected patterns.
- An RGB camera: works as an ordinary colour camera.

The Kinect computes depth with a triangulation process.[3] After a calibration procedure, both the RGB image and the IR depth data can be fused to form a coloured 3D point cloud of about 300,000 points in every frame. Like any camera, Kinect cannot cover the entire continuous resolution of the physical scene [18]. It actually projects the captured disparity map on a set of discrete parallel planes. Consequently, some data, which should be positioned between the planes (according to their real world continuous (x, y, z) coordinates), are either lost or shifted to the closest available discrete range level. The accuracy of the sensor is largely affected by this low-resolution-related

[3] http://wiki.ros.org/kinect_calibration/technical, 2014.

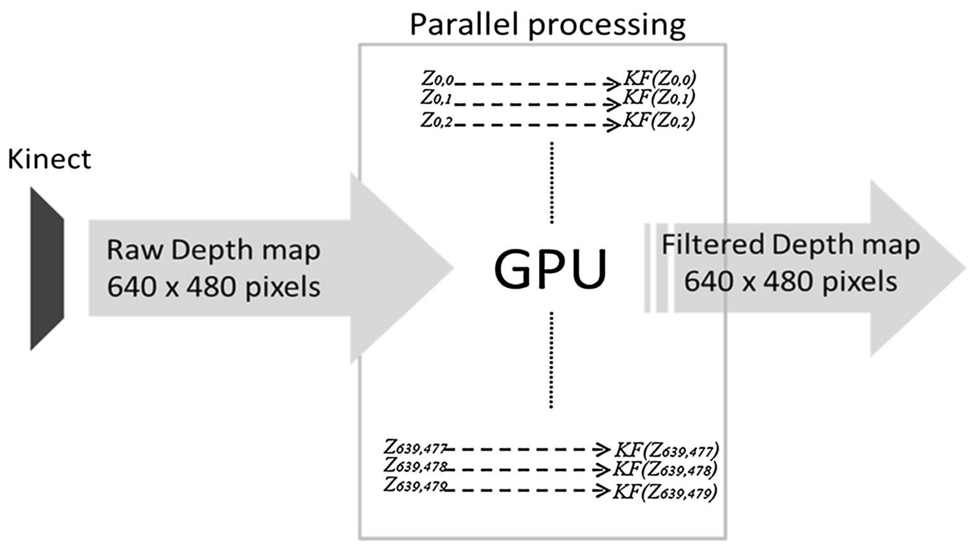


Fig. 3 Real-time RGBD data filtering architecture

behaviour. More importantly, if we use Kinect for applications that require fairly accurate 3D pose data, this drawback will adversely affect the measurements and will result in erroneous sensing for the whole system. The innovative point in our research work is that, inspired by the 3D structure of the sensor's output, and using a well-established filtering scheme [19], we were able to improve the capabilities of RGBD sensors and to increase the robustness of the measurements over time. However, the adaptation of the filter to the sensor's output is not possible before the fulfilment of a number of conditions, such as the assumption of Gaussian white noise and that the quantity that we wish to optimise (depth measurements) should evolve linearly over time. In the following section, we present the Kalman filter. Then, we develop a thorough analysis followed by the proof that all the conditions required to apply the Kalman filtering scheme are satisfied with Kinect data.

5 Kalman filter

The Kinect has been designed for gaming applications in which the user is relatively close to the sensor at all times. Professional applications using its data can only be accurate at a close range. Indeed, for depth values greater than 3.50 m the error can reach ±20 cm. However, this inaccuracy is unacceptable in most end applications.

To overcome the low quality of data, many filters exist in the literature, but up to now, none has perfectly adapted to work on RGBD sensors. In this study, we apply the Kalman filter to accurately stabilise the Kinect data capture over time. We also improve its capabilities to cover a larger view. This work is motivated by some properties we discovered concerning the depth data that permit us to fit it into a Kalman filter framework. The working principle of this filter is based on a recursive prediction of the next state and its correction. The Kalman filter is distinguished by its ability to run in real time, using only the present measurements as input and the previously estimated state, with no additional history of the system's behaviour [20].

The general form of state-transition filters to predict or estimate the state of a dynamic system from a series of incomplete or noisy measurements is defined by the following equations:

$$x_t = Ax_{t-1} + Bu_t + w_t, \tag{1}$$

$$z_t = Hx_t + v_t. \tag{2}$$

Linear Kalman filter equations in prediction–correction form are given below:

Prediction:

$$\overline{x_t} = Ax_{t-1} + Bu_t + w_t, \tag{3}$$

$$\overline{P_t} = AP_{t-1}A^T + Q_t. \tag{4}$$

Correction:

$$K_t = \overline{P_t}H^T(H\overline{P_t}H^T + R_k)^{-1}, \tag{5}$$

$$x_t = \overline{x_t} + K_t(z_t - H\overline{x_t}), \tag{6}$$

$$P_t = (I - K_tH)\overline{P_t}, \tag{7}$$

where for each discrete time step, t, x_t is the state variable; $\overline{x_t}$ is the state estimate; z_t is the measured entity; P_t is the state error covariance; $\overline{P_t}$ is the a priori estimate of the state error covariance; K_t is the Kalman gain; A is the state-transition model; B is the control-input model; H is the observation model; Q_t is the covariance matrix of process noise; R_k is the covariance of measurement noise of the kth Z-Level; w_t is process noise; v_k is measurement noise at the

kth Z-Level; u_t is the control signal. Process and observation noise models should be independent, white and follow a normal distribution: $w_t \sim N(0, Q_t)$; $v_k \sim N(0, R_k)$.

To adapt Kalman filter to our specific case, we should first demonstrate that the sensor's model and its data satisfy the requirement to fit into Eqs. 3, 4, 5, 6, 7. During the experiments, we found that Kinect RGBD pixels lie in parallel planes towards the positive Z direction. The depth values are limited to a known set of levels. To enhance the accuracy of the sensor, without any extra hardware, we proposed an adaptation to the Kalman filter, which over a small number of frames associates each pixel with an optimal depth value.

6 Kalman filter on Kinect's data

6.1 Z-resolution

To study the nature of depth resolution of the camera, we pointed the camera parallel to a large flat wall as shown in Fig. 4a. This setup allows us to capture a cloud of points from the whole Kinect operating range. We also aim to characterise the necessary parameters for the filter.

As shown in Fig. 4b, depth resolution is inversely proportional to the distance from the sensor. In addition, the points in the capture (taken from the same frame) are distributed in independent clusters that we define as "Z-Levels". Accordingly, we formulate Kinect's data as a finite set of points distributed on parallel planes (Z-Levels), where every plane constitutes a partition of the whole capture.

The mathematical definition is as follows:

K : Set of indices ranking the parallel planes.

I: Set of indices indexing the points lying in the planes.

C: set of the whole point cloud data (Fig. 4b).

$Z_k, k \in K$: Plane in $\boldsymbol{C}$ (Fig. 4c).

$P_i(x_i, y_i, z_i)$, $i \in I$: Point in RGBD space, lying in a given Z-Level, $Z_k = z_i$, $k \in K$, (Fig. 4c).

Every point cloud C satisfies the properties:

$$C = U_k Z_k, \quad k \in K \tag{8}$$

$$\forall Z_{k1}, Z_{k2} \in C, k_1, k_2 \in K, k_1 \neq k_2; Z_{k1} \cap Z_{k2} = \emptyset \tag{9}$$

$$\forall p_i \in C, i \in I, k \in K; \exists! Z_k, p_i \epsilon Z_k \tag{10}$$

$$\forall Z_k \in C, k \in K; Z_k \perp Z_{axis}. \tag{11}$$

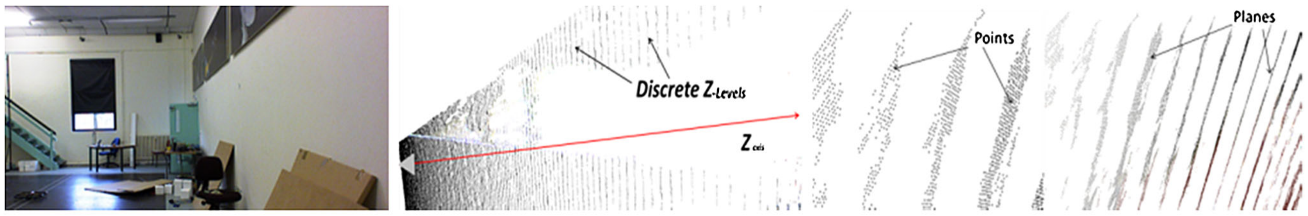
6.2 Depth noise statistics

The Kinect, as an electronic device, has hardware-related noise. This noise is caused by reference template accuracy, calibration process, lighting conditions and the objects' surface properties [21]. Errors in the projected data increase with increasing Z distances, because of the decrease in depth resolution Fig. 4b. A study that we conducted to discover the nature of noise affecting the depth measurements showed that it has a Gaussian distribution with varying standard deviations. These depend on the range separating the sensor and the scene. Figure 5a shows some samples that were taken at 3.406 m from the camera. Based on the graph, we can extract the corresponding standard deviation, which was 0.075 m.

When we re-projected the sampled points back to their original depth map, we found that the standard deviations σ_k could be formulated by the equation:

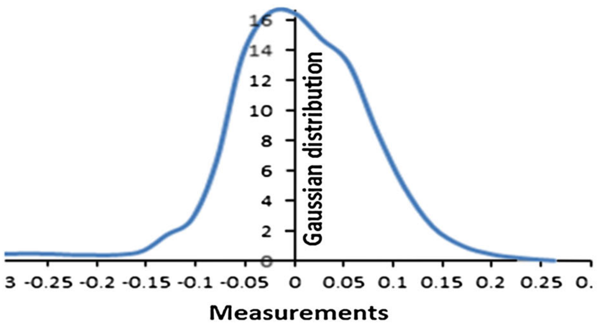
$$\sigma_k = (Z_{k+h} - Z_{k-h})/2, \quad \forall k \in K, h \in \mathbb{N} \tag{12}$$

where σ_k is the average distance between the two extremities of the $2h+1$ Z-Levels and the central one Z_k to which the sampled point belongs (Fig. 5b). As a result, at every level Z_k, the Kinect noise remains Gaussian and σ_k is its standard deviation Eq. 12. Empirically, the best results were reached with $h = 3$. This property allows us to prove the Gaussian nature of the quantisation noise affecting depth data, and to consequently satisfy the first condition required to apply the Kalman filter. Moreover, we can precisely attribute a standard deviation σ_k to every Z-Level, which will serve to compute optimal depth measurements for all the points in the cloud.

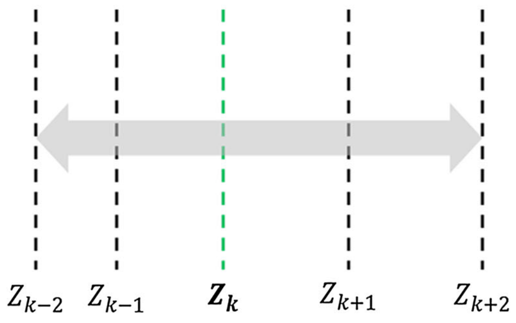


(a) Setup's RGB output (b) Setup's depth output (c) Point cloud components

Fig. 4 Kinect's point cloud structure



(a) Kinect measurement noise distribution at 3.406m



(b) Deviation range *5 Z-Levels*

Fig. 5 Z data quantisation noise behaviour and statistics

6.3 States of an IR pixel

When we point the sensor against a static scene, where both the sensor and the scene remain steady during the whole capture, and we observe the depth map over time Fig. 6a, we notice fluctuations in almost 90 % of all the map's elements (the range reading at every pixel of the depth map). The change is limited to a finite set of values close to each other. If we closely observe the rendered points, we note that they tend to disappear from one Z-Level and reappear in a neighbouring one as illustrated in (b). In addition, for every capture of any scene, there is a finite set of repetitive depth values. In other words, we can predict the possible discrete depth values that we may encounter in the output data. As explained above, the Kinect sensor works in a discrete set of depth elements that we call "Z-Levels". Every level constitutes a partition of the whole set of points within a frame (Eqs. 8, 9, 10), and has the property of being completely independent from the neighbouring levels and orthogonal to Z-axis (Eq. 11). As a result, we cannot find a point out of these parallel planes. This is what really appears in all the captured data if we rotate the scene over the X-axis or Y-axis (the points lie in planes parallel to XY). The importance of such information for Kinect-based applications is that we can study the relationship between depth measurements taken over different frames. That is, if we find that two successive depth measures are related to a linear function, we will have fulfilled the second condition required to use the Kalman filter.

When the sensor is stationary, the depth map keeps changing because of points jumping from one Z-Level to another Fig. 6b, not necessarily adjacent, but within a limited radius Eq. (12).

When the points change their depth level, the 2D $(u_i,\ v_i)$ image coordinates on the screen remain the same, but their world coordinates $(x_i,\ y_i,\ z_i)$ change. This is true because every point $P_i\,(x_i,\ y_i, z_i)$ in the 3D world lies on a line from the centre of the camera passing through the pixel $(u_i,\ v_i)$ on the screen towards the scene (Fig. 6b), following the direction of the perspective frustum [22]. Using this information, we can infer the relationship between two successively measured 3D coordinates.

From the intrinsic parameters of the camera $(f_x,\ f_y;\ c_x,\ c_y)$, we have the equations:

$$\begin{cases} u_i = (f_x/z_i)x_i + c_x \\ v_i = (f_y/z_i)y_i + c_y \end{cases} \quad z_i \neq 0. \tag{13}$$

As pixel coordinates in two successive frames remain the same, from Eq. 13 we get with $z_i \neq 0, z_i^+ \neq 0$ [(+) means the new coordinates in the following frame]:

$$\begin{cases} u_i^+ = u_i \longrightarrow \text{yields}(f_x/z_i^+)x_i^+ + c_x = (f_x/z_i)x_i + c_x \\ v_i^+ = v_i \longrightarrow \text{yields}(f_y/z_i^+)y_i^+ + c_y = (f_y/z_i)y_i + c_y \end{cases}. \tag{14}$$

Finally, after simplification of Eq. 14, we get:

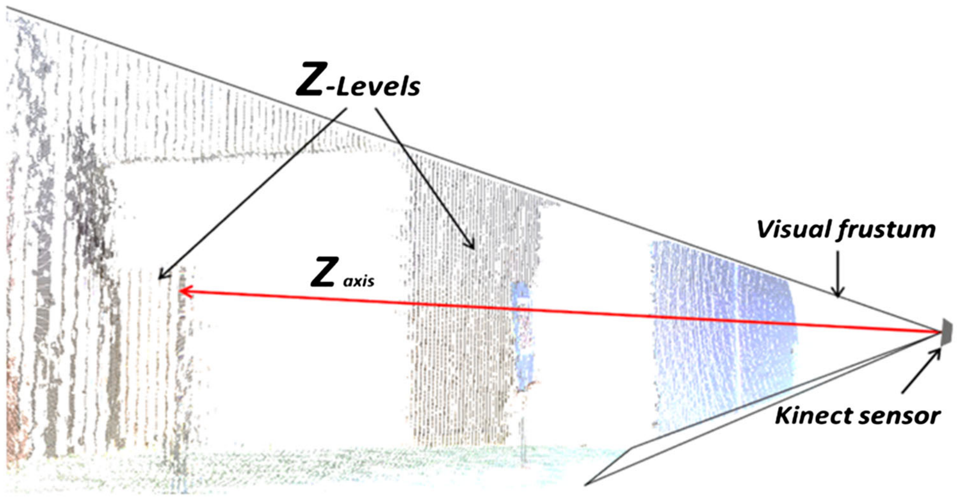
$$\begin{cases} x_i^+ = \left(\dfrac{z_i^+}{z_i}\right)x_i \\ y_i^+ = \left(\dfrac{z_i^+}{z_i}\right)y_i \end{cases}. \tag{15}$$

Equation 15 proves the linearity between points projected at the same pixel on the screen over different frames. Adding this to the Gaussian nature of noise, we can safely use the Kalman filter as a real-time filtering tool for RGBD sensors. The effect of the filter can clearly be seen in Figs. 7, 8.

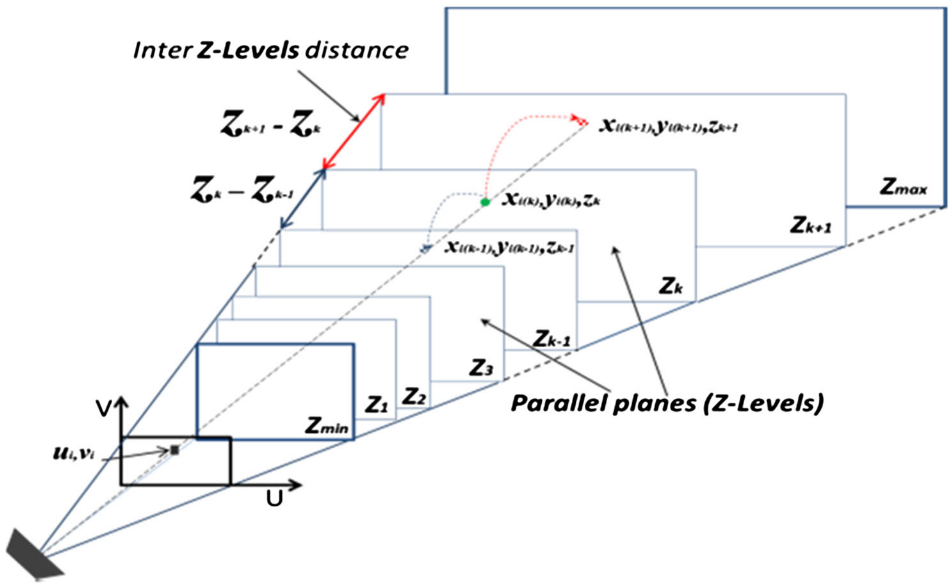
6.4 The Kalman filter adaptation to the Kinect sensor

For a given pixel in the frame at time step t, $\bar{Z}_t$ is the state estimate; $\tilde{Z}_t$ is the measured depth; $\bar{P}_t$ is the a priori estimate error covariance and K_t is the Kalman gain. We have one-to-one scalar correspondence between state/measurements, so $H = 1. A = 1$ as the depth should not change beyond the magnitude of noise between two successive

Fig. 6 Z-Levels' properties



(a) Z-levels' structure



(b) linear displacement of a given point towards the perspective frustum lines

frames (σ_k). $B = 0$ for a fixed sensor. There is no process noise, $Q \approx 0$ (we assume that the system is precisely modelled). $R_k = \sigma_k^2$ is the covariance of observation noise (σ_k is the standard deviation which describes the magnitude of noise around its mean $\bar{Z}_t$ and differs from one Z-Level to another proportionally to the distance from the sensor). For a static sensor/dynamic scene setup, Kalman Eqs. 3, 4, 5, 6, 7 become:

Prediction:

$$\bar{Z}_t = Z_{t-1}, \tag{16}$$

$$\bar{P}_t = P_{t-1}. \tag{17}$$

Correction:

$$K_t = \bar{P}_t(\bar{P}_t + R_k)^{-1}, \tag{18}$$

$$Z_t = \bar{Z}_t + K_t(\widetilde{Z}_t - \bar{Z}_t), \tag{19}$$

$$P_t = (1 - K_t)\,\bar{P}_t. \tag{20}$$

From the equations above, one may think that at a given pixel, $\bar{Z}_t$ does not evolve over time, although important variations in depth occur if the observed object moves backward or forward. Our filter optimises the estimates of depth under the assumption that the dynamics of the scene slightly change between two successive frames (the change

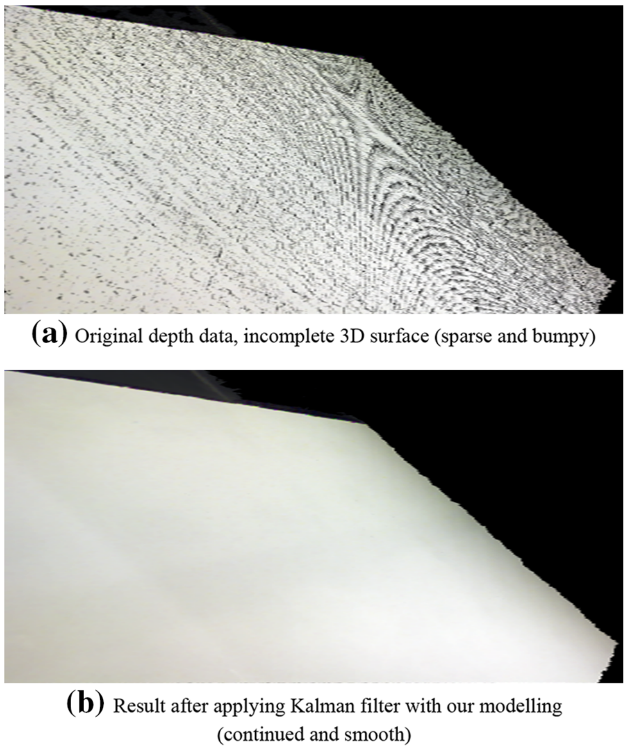

(a) Original depth data, incomplete 3D surface (sparse and bumpy)
(b) Result after applying Kalman filter with our modelling (continued and smooth)

Fig. 7 The effect of Kalman filter on point cloud data

is not abrupt $\Delta Z \leq \alpha \times \sigma_k$, within 33 ms; 33 ms = 1/30 fps). The parameter α is determined empirically. We found that $\alpha = 2.5$ gives the best results. In practice, our customisation of the Kalman filter is generally verified, because we deal with physical objects that move gradually and continually. The elementary displacements of these entities are small given the high frame rate of capture that we are ensuring by implementing the filter on the GPU. Based on this, when the depth reading changes its levels, the filter updates the pixel's workspace ($\bar{Z}_t$, R_k, $\bar{P}_t$, K_t). However, when the difference between two successive z values becomes greater than 2.5 $\times$ σ_k, the workspace is reinitialised according to the current value of depth (the object jumps from Z-Level k to Z-Level' $\bar{\boldsymbol{Z}}_{t+1} = \check{\boldsymbol{Z}}_{t+1}$, $R_{k'} = 2.5 \times \sigma_{k'}$, $\bar{P}_{t+1} = R_{k'}$, $K_{t+1} = \bar{P}_{t+1}(\bar{P}_{t+1} + R_{k'})^{-1}$). As a result, the steadier the scene is, the better the filter performs. In real scenarios, most of the pixels within the image do not change beyond the threshold of every frame. This fact helps the filter to operate smoothly in an indoor environment (Kinect is designed to work only indoors) where the scene does not abruptly change all the time.

7 Kalman filter effect on RGBD data for moving vehicles' tracking

Tracking applications are very concerned with the accuracy in position of the tracked entity. However, Kinect raw data are not accurate enough to precisely localise an object within its neighbourhood. When the sensor captures a point cloud, the 3D data are automatically distributed on the discrete Z-Levels Fig. 9b. However, the original point data come from the continuous real world. Their corresponding images in Kinect's space lie in the sensor's parallel planes. The error in measurement is proportional to the gap between Z-Levels where the points are projected. Nonetheless, the Kalman filter takes these noisy raw data as input, optimises them, and consequently approaches the best of their real-world counterparts Fig. 9c. The effect of the filter can be clearly seen in Fig. 10. The two images in Fig. 10a, b depict the raw and the filtered trajectories, respectively, for a moving robot captured by the same Kinect camera. The blue points in Fig. 10a represents the raw positions taken by the robot. If we observe closely the deeper points (greater z_i), the gaps between the parallel Z-Levels are clearer (because of the drop in resolution as we get far away from the camera). However, the Kalman filter's smoothing effect as seen in Fig. 10b optimally condenses the sparse and discrete Z_k around their corresponding real-world true positions $\bar{Z}_t$. This behaviour reduces the error in the 3D point cloud. Accordingly, the position of the tracked object becomes more precise and reliable.

8 Kalman filter effect on RGBD data for depth image registration

Image registration is necessary to reconstruct 3D models of real objects for simulation, and virtual and augmented reality applications. Feature extraction and matching are the basic tools to find the respective correspondences between two different images before aligning them. Figure 11b illustrates the captured point data which are distributed on parallel planes (as we explained earlier). When the same real-world feature is detected in two different point clouds, we find it lying in a given Z-Level that is naturally discrete and does not reflect the true position of the feature in the scene. Consequently, the whole 3D data of the scene are discretised because of the sensor's nature. As a result, equivalent features within different point clouds will be wrongly matched. Hence, alignment error increases (Fig. 11d, e). On the other hand, the application of the Kalman filter optimises the positions of the tracked features and produces the best possible result, given hardware limitations of the camera. The Kalman filter optimally places the discrete points in the continuous real space (off the discrete Z-Levels) (Fig. 11c). Therefore, the features obtained from different viewpoints should have closer 3D geometric properties and the subsequent registration is achieved with less error (Fig. 11f). The Kalman filter is more useful at greater ranges, because Kinect accurately measures the depth at small ranges (below 1.5 m).

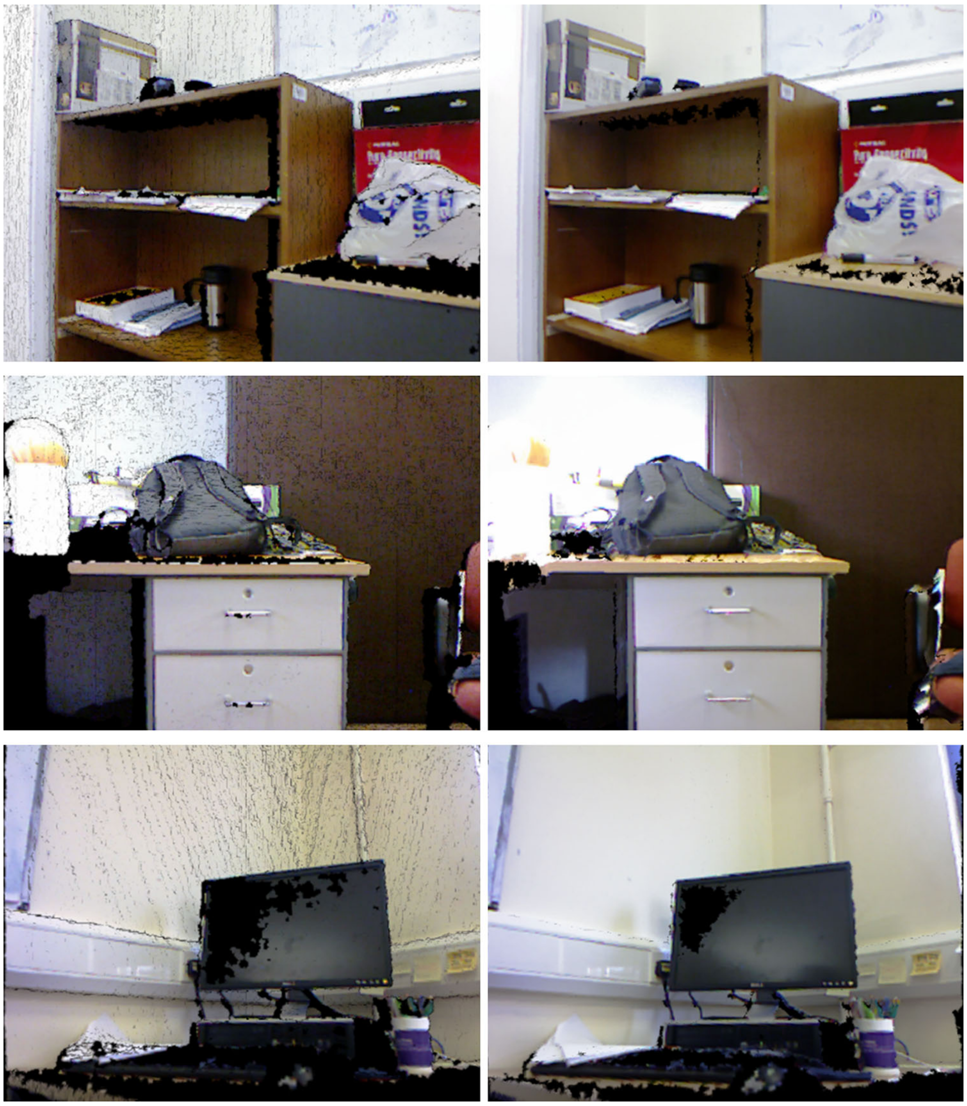


Fig. 8 On the left, raw 3D point clouds; and on the right, the corresponding filtered point clouds (Kalman filter)

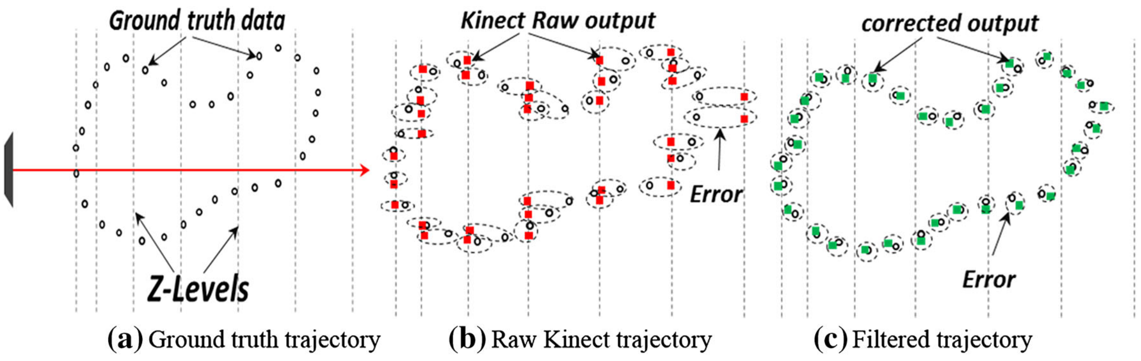
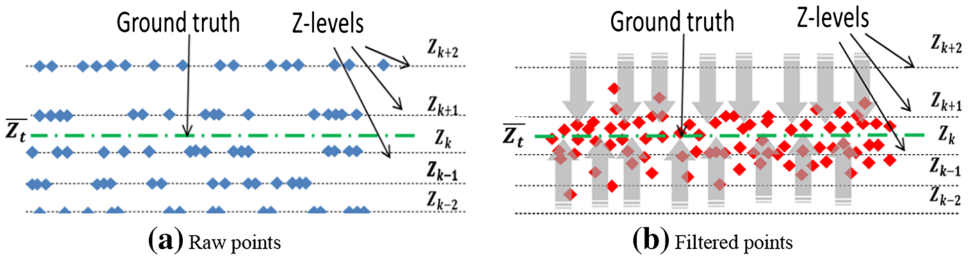


Fig. 9 Kalman effect on Kinect's data for object tracking applications

Fig. 10 Kalman effect on position data



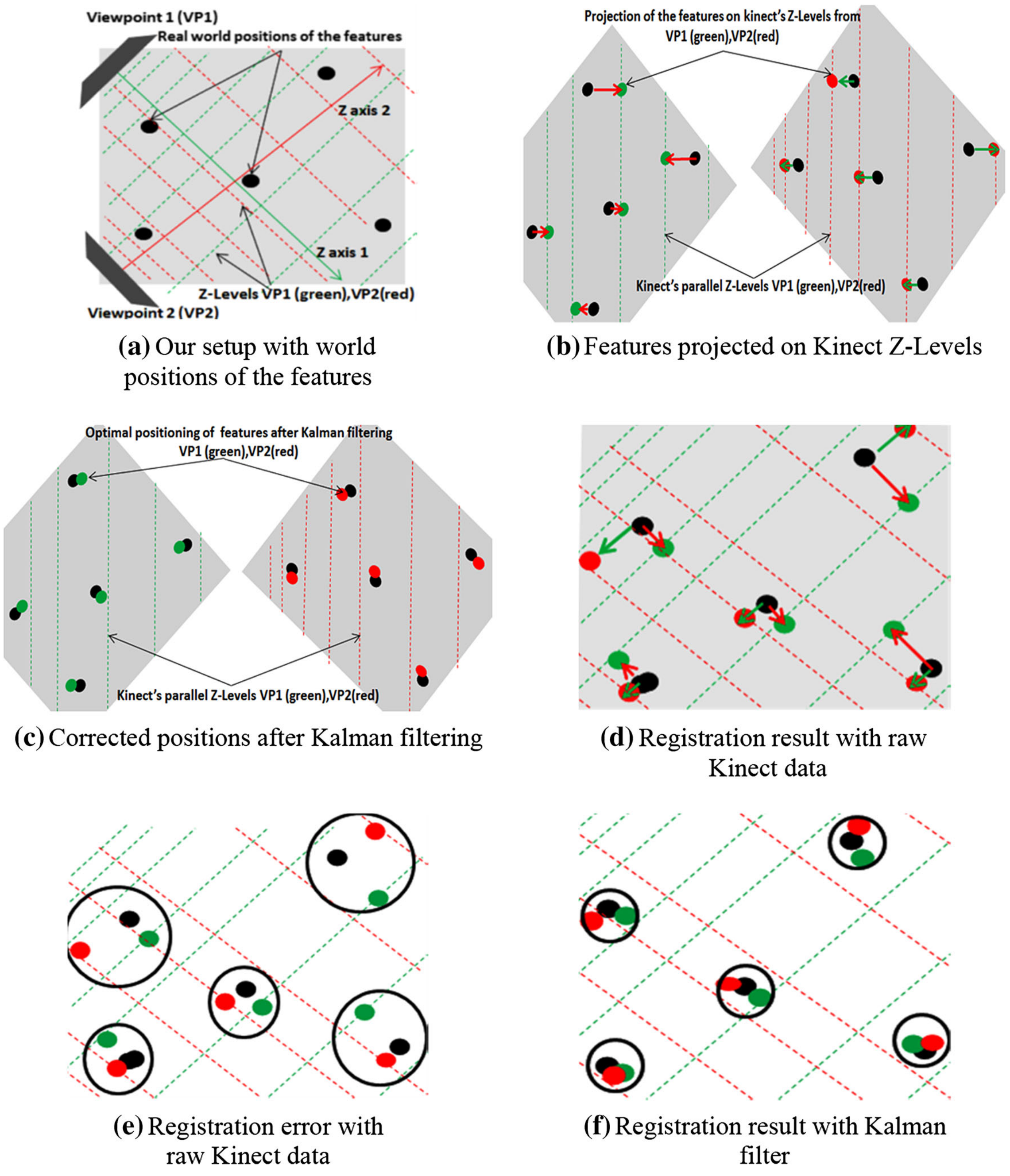
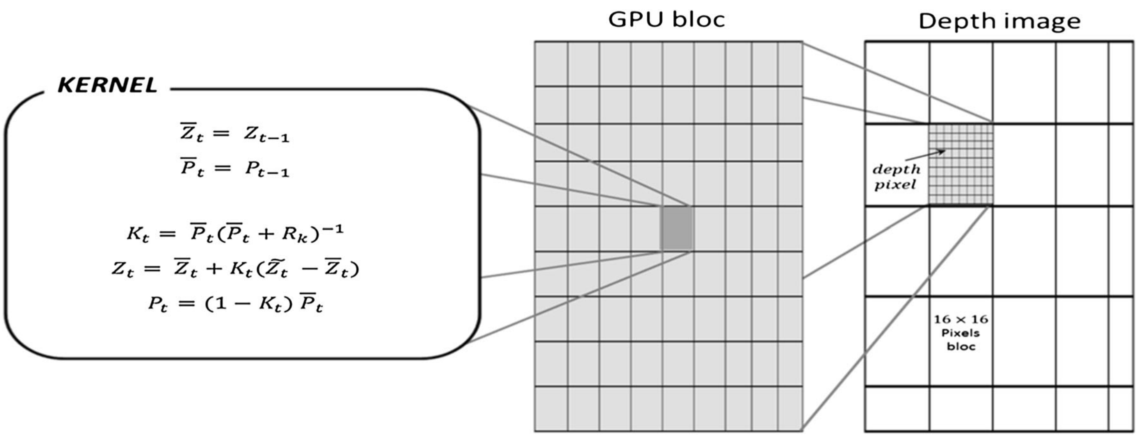
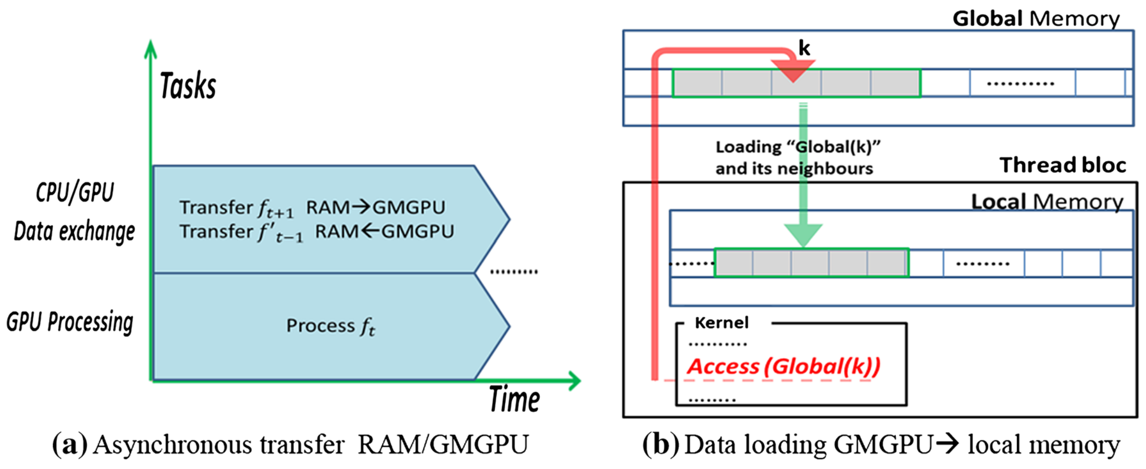

Viewpoint 1 (VP1)
Real world positions of the features
Z axis 2
Z axis 1
Z-Levels VP1 (green),VP2(red)
Viewpoint 2 (VP2)
(a) Our setup with world positions of the features
Projection of the features on kinect's Z-Levels from VP1 (green),VP2(red)
Kinect's parallel Z-Levels VP1 (green),VP2(red)
(b) Features projected on Kinect Z-Levels
Optimal positioning of features after Kalman filtering VP1 (green),VP2(red)
Kinect's parallel Z-Levels VP1 (green),VP2(red)
(c) Corrected positions after Kalman filtering
(d) Registration result with raw Kinect data
(e) Registration error with raw Kinect data
(f) Registration result with Kalman filter

Fig. 11 Effect of Kalman filtering on image features



KERNEL
$\overline{Z}_t = Z_{t-1}$
$\overline{P}_t = P_{t-1}$
$K_t = \overline{P}_t(\overline{P}_t + R_k)^{-1}$
$Z_t = \overline{Z}_t + K_t(\widetilde{Z}_t - \overline{Z}_t)$
$P_t = (1 - K_t)\overline{P}_t$
GPU bloc
Depth image
depth pixel
16 × 16 Pixels bloc

Fig. 12 Kalman filter GPU implementation for depth map filtering



Tasks
CPU/GPU Data exchange
GPU Processing
Transfer f_{t+1} RAM→GMGPU
Transfer f'_{t-1} RAM←GMGPU
Process f_t
Time
(a) Asynchronous transfer RAM/GMGPU
Global Memory
k
Loading "Global(k)" and its neighbours
Thread bloc
Local Memory
Kernel
Access (Global(k))
(b) Data loading GMGPU→ local memory

Fig. 13 Optimisation of data exchange in the GPU

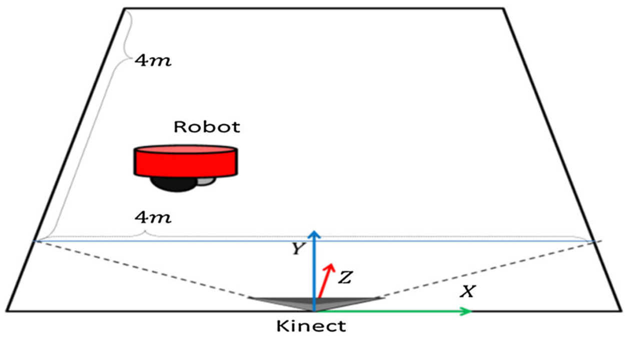


Fig. 14 Robot tracking experimental setup

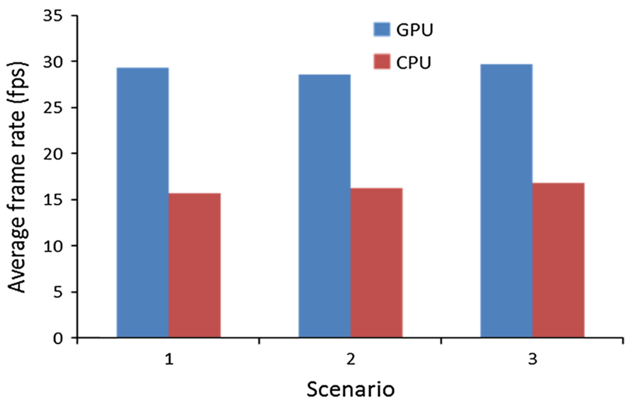
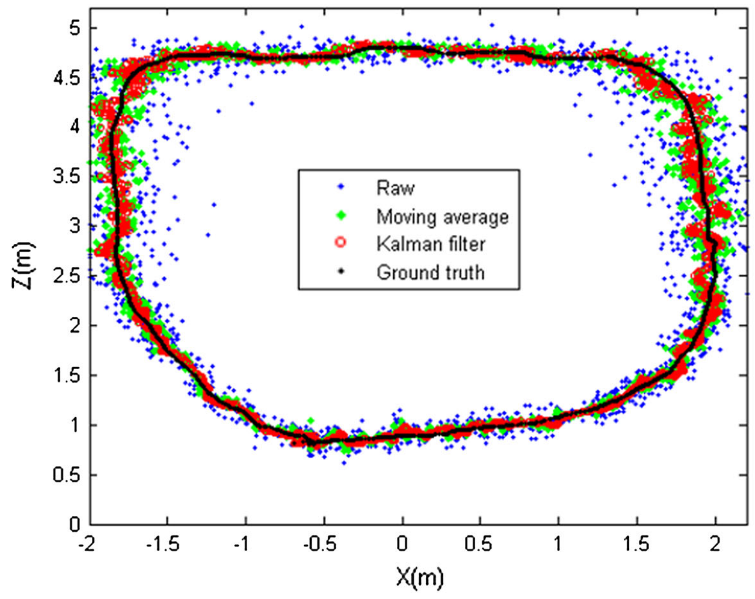


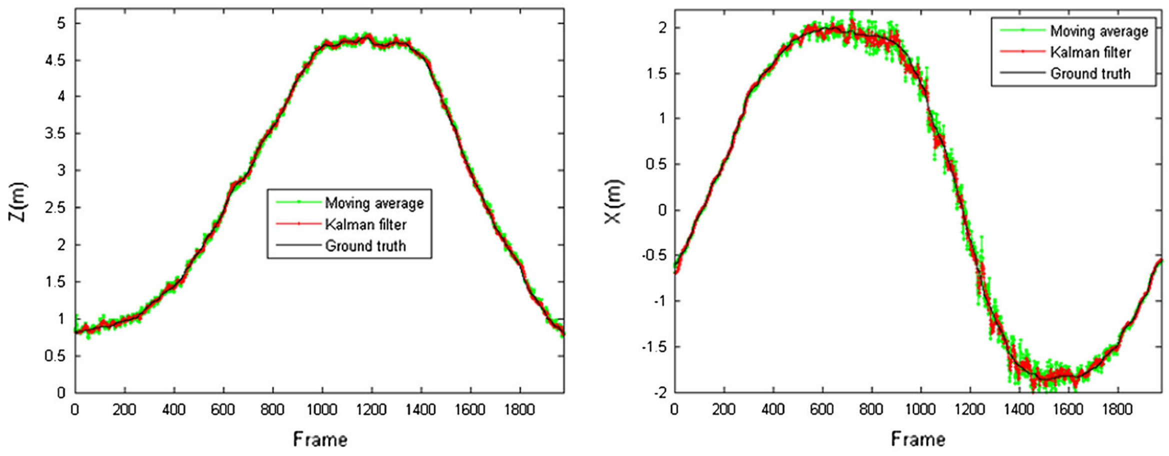
Fig. 15 CPU/GPU benchmarking of KF for Kinect

9 Results and discussion

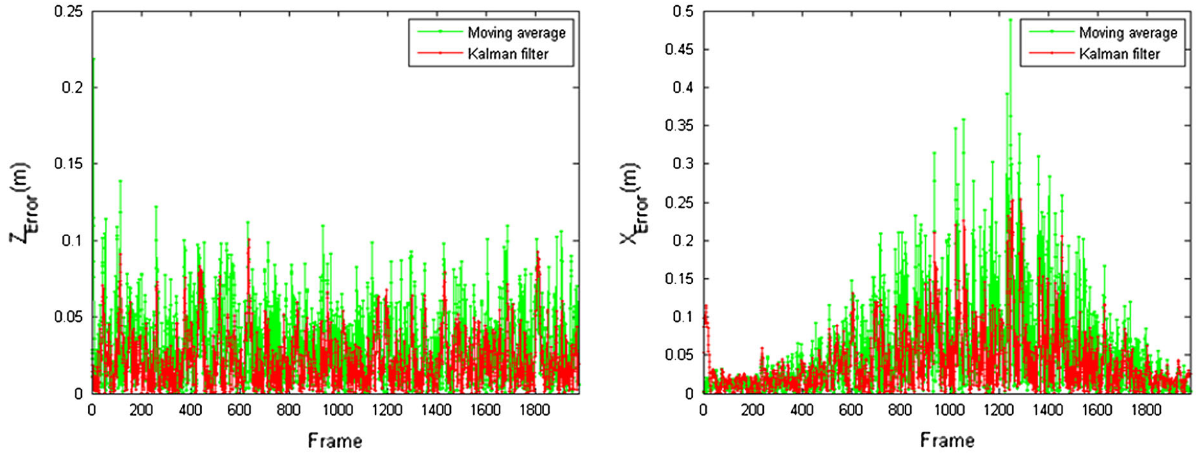
All the following stages are based on our hardware configuration which includes an INTEL i7 3,930 K CPU with six physical cores (two logical cores per physical) running at 3.20 GHz, 16.0 GB of RAM, along with an NVIDIA GeForce 2 GB GTX 680 GPU. All the programing in our work was done in C++ for the CPU and CUDA for the GPU.



(a) Trajectories of the vehicle, raw in blue, KF in red, MovAve in green and the ground truth in black

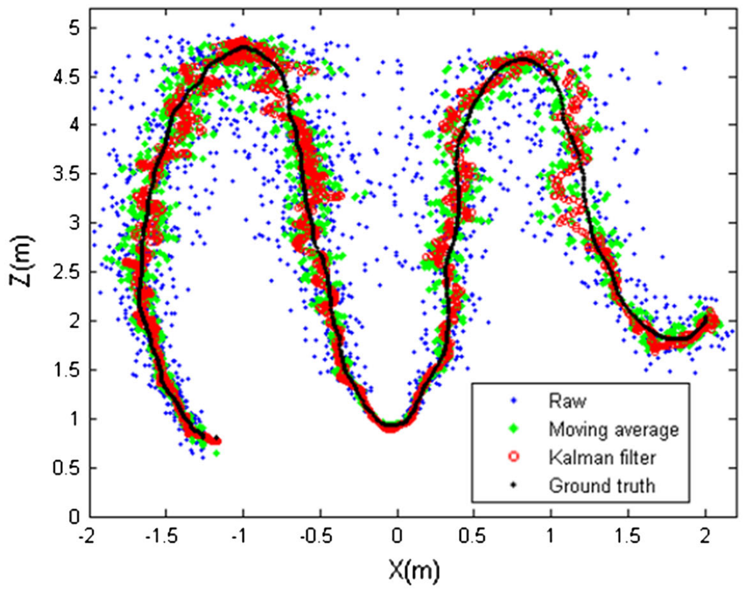


(b) Z and X graphs , KF in red, MovAve in green and ground truth in black

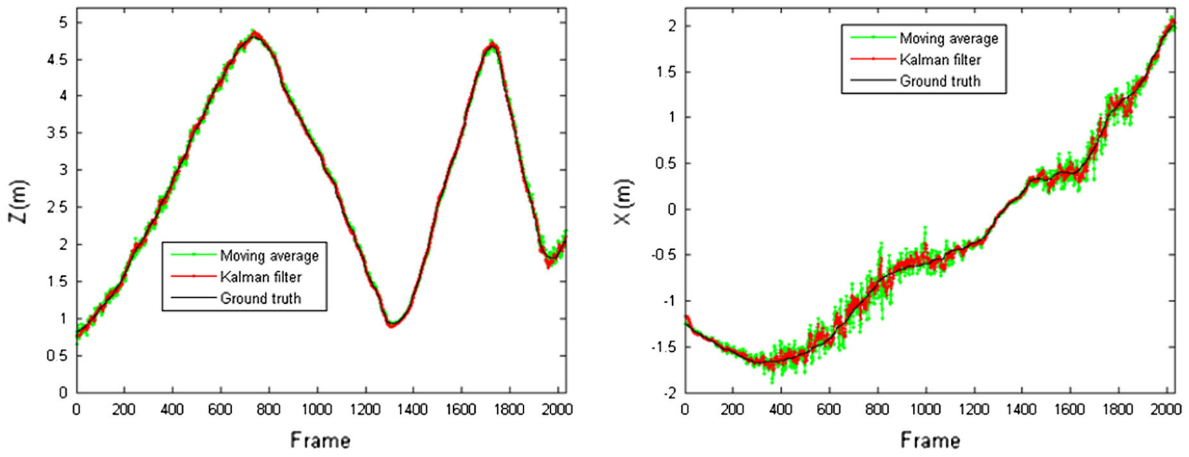


(c) Z and X error graphs , KF in red, MovAve in green

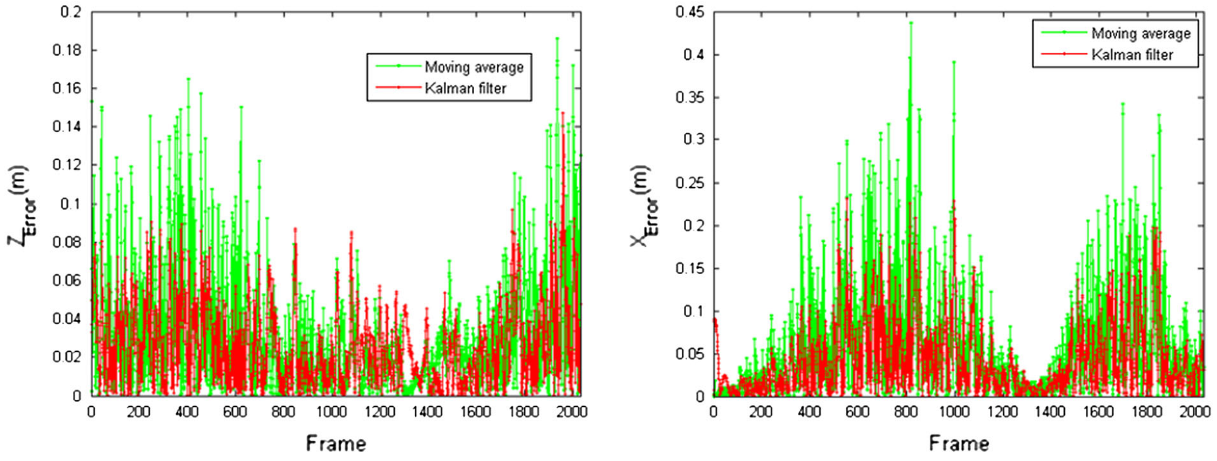
Fig. 16 Tracking scenario 1 (*circle*)



(a) Trajectories of the vehicle, raw in blue, KF in red, MovAve in green and the ground truth in black

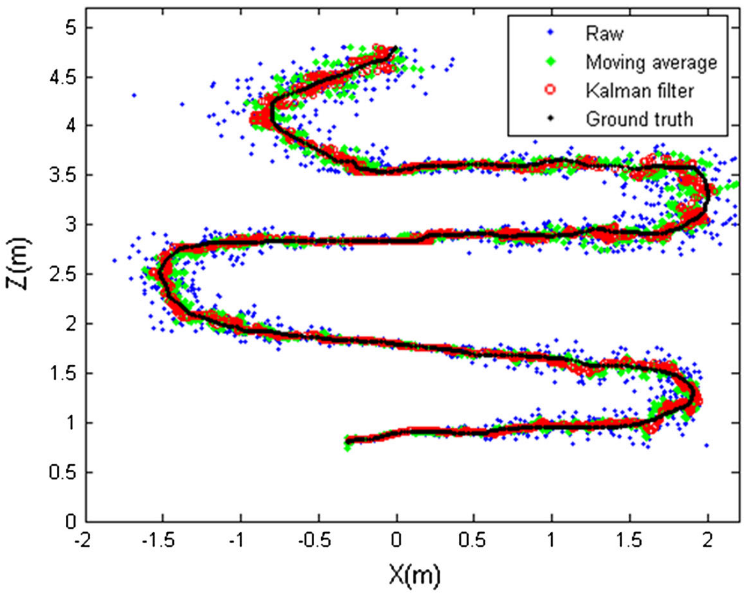


(b) Z and X graphs , KF in red, MovAve in green and ground truth in black

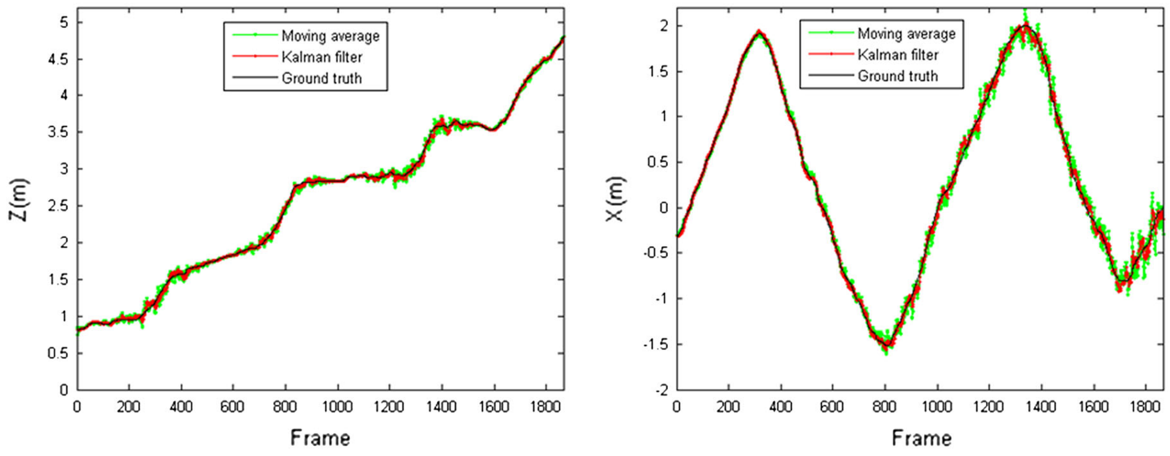


(c) Z and X error graphs , KF in red, MovAve in green

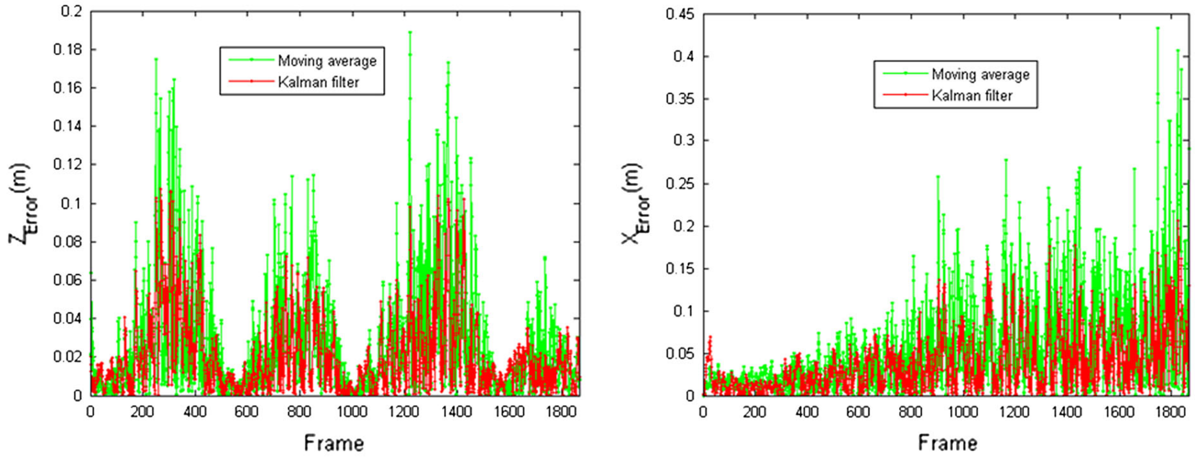
Fig. 17 Tracking scenario 2 (*left to right*)



(a) Trajectories of the vehicle, raw in blue, KF in red, MovAve in green and the ground truth in black



(b) Z and X graphs , KF in red, MovAve in green and ground truth in black



(c) Z and X error graphs , KF in red, MovAve in green

Fig. 18 Tracking scenario 3 (*front to back*)

Table 1 RMSE results of our tracking scenarios

Scenarios	Components	MovAve (m)	KF (m)	Difference (m) (MovAve–KF)
Scenario 1 (circle)	Z	0.040307	0.028102	0.012205
	X	0.081796	0.056366	0.025429
Scenario 2 (left-to-right)	Z	0.040307	0.028102	0.012875
	X	0.081796	0.056366	0.030445
Scenario 3 (front-to-back)	Z	0.045275	0.033223	0.012053
	X	0.08948	0.058883	0.030597

9.1 GPU implementation of the Kalman filter for depth map filtering

The objective of the present work is to filter VGA resolution (640 × 480) depth images at the same frame rate of the camera to allow the following applications to fully exploit the frame rate offered by the sensor (30 fps). When we first ran the filter on regular CPU, the maximum reached frame rate was 17 fps. Hence, the need to implement the solution on the GPU emerged.

Image data are more naturally organised to fit GPU blocs where every element in the bloc (kernel) processes a single pixel at a time [23]. Figure 12 illustrates how the depth image output by the camera is divided into image blocs of a constant size of (16 × 16 pixels; so 256 threads is the size in of the bloc in our implementation). The pixels of the same image bloc are processed simultaneously in the same GPU thread bloc. As a result, for every pixel in the image, a kernel in the GPU is attributed. The latter runs the actual filtering (KF) on a single depth pixel (range reading) and saves the necessary data for the next frame (Z_t, P_t). This scheme is straightforward because there are no constraints between the pixels and the order in which they should be processed. Otherwise, more specific techniques should be applied to benefit from the parallel computing ability of the GPUs. The complexity of processing is reduced to the complexity of the algorithm running in the kernel (Kalman filter), which indeed is constant.

Other methods should be addressed to optimise the utilisation of all the available hardware capability. Basically, the design of heterogeneous algorithms aims at higher occupancy of the processors and a full usage of the bandwidth when exchanging data between the central memory (RAM) and the global memory of the GPU (GMGPU) [24]. To this end, we focus on two optimisation aspects:

9.1.1 *Running asynchronous transfers*

When the GPU is processing the current frame, the bus between it and the central memory is entirely free. We therefore benefit from this idleness in exchanging data. In other words, the following frame (f_{t+1}) is sent from the RAM to the GMGPU, and the already available result (f'_{t-1}) is sent back to the RAM. Simultaneously, the current frame (f_t) is being processed on the device (GPU) Fig. 13(a).

9.1.2 *Memory coalescing*

The GPU automatically loads the content of adjacent memory cells because its internal design assumes that it is highly probable for neighbouring data within the same area to be sooner requested as well [25]. Memory coalescing is another optimisation that significantly helps to increase the probability of threads in same warp (a group of 32 threads from the same thread bloc running simultaneously) to fetch data from the memory together. The purpose of memory coalescing is to ensure that the threads fetch data within the same memory segment to only pay one memory transaction. However, if the threads of the same warp fetch sparse addresses, then it will cost 32 memory transactions.

Appropriately organising the data in device memory allows such contiguous access to automatically happen. Programmatically, Structure of Arrays rather than the easy to use Array of Structure significantly increases the chances of loading a chunk of memory containing the data for not only the thread which requests it, but also for its neighbours in the warp. Figure 13b illustrates what happens when a thread fetches a given cell in the global memory.

9.2 Object tracking applications

To validate our findings about the filtering effect of the Kalman scheme on the Kinect data, we conducted an experiment in which a moving vehicle was tracked by one Kinect camera Fig. 14. The robot moves in a closed space of 4 m × 4 m, and the aim is to find the global position of the robot within the surrounding environment at the same time of the capture.

We tested our solution against Moving Average filter (MovAve) [26] to justify the rationale of fitting the Kalman equations to the Kinect sensor and to show the proof of validity.

The two important components to localise the robot are Z and X coordinates (Y is almost unchangeable over time for ground robots. It can be included without any restriction). The purpose of this approach is to test the accuracy of Kinect in issuing a 3D special positions for a given object in real time. The filter affects only the depth data (Z component in this setup). Hence, the computation of the two other coordinates is based on Z and the calibration parameters of the IR camera Eq. 13).

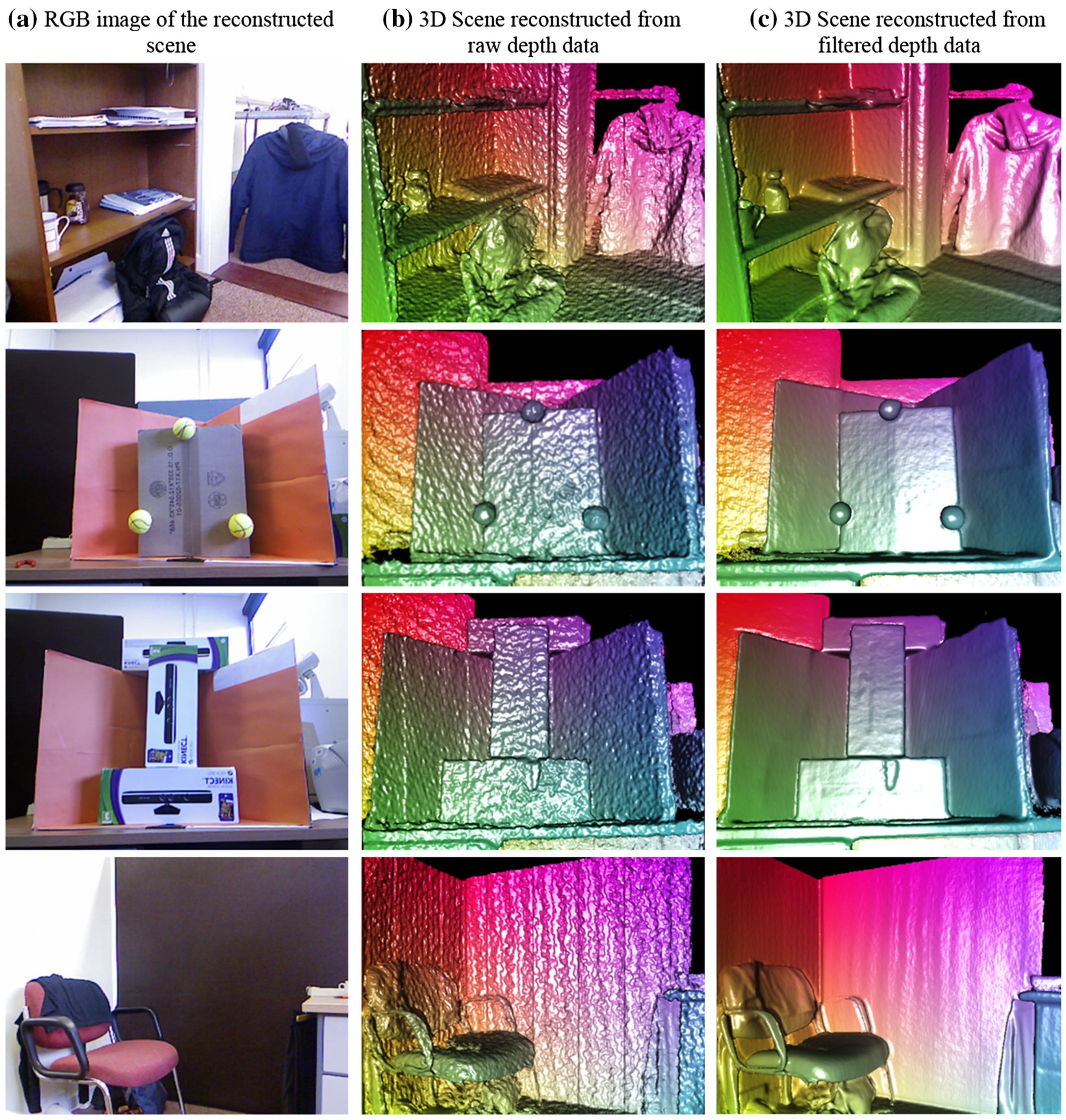

(a) RGB image of the reconstructed scene
(b) 3D Scene reconstructed from raw depth data
(c) 3D Scene reconstructed from filtered depth data

Fig. 19 Experimental results for 3D reconstruction applications

Figure 15 illustrates our GPU/CPU benchmarking for the three tracking scenarios. The outcome of the GPU implementation is clearly seen. The whole frame rate of tracking is just below 30 fps (almost one frame processed every 33 ms).

For generality, we tested our tracker on three different scenarios where the robots were moving in front of the camera in circular motion, left to right (swinging back and forth) and front to back (swinging left and right). For every scenario, we plotted the ground truth trajectory taken by the vehicle, along with the raw position and the filtered trajectories resulting from the Kalman filter (KF) and the moving average filter (MovAve).

To assess the accuracy of our approach, we evaluated the root mean square error (RMSE) for both Z and X. The results can be seen in Figs. 16, 17, 18.

In all the three scenarios, it is clearly shown that Z and X graphs filtered with KF are impinged with less error because of KF optimal smoothing effect. Moreover, the further the robot gets from the camera to the upper left and right corners, the higher is the error. Noisy fragments of the trajectory correspond to the peaks of X and Z error plots. Ultimately, the filtered trajectory (in red) is always the closest to the ground truth (in black). Its RMSE is also smaller than that of MovAve. In other words, at every position KF-RMSE is 1.0 cm less than MovAve-RMSE Table 1. The raw data (blue) is sparser and unsteadier. Its corresponding error is higher.

9.3 Registration applications

Unlike tracking, registration applications use many features to find the correct mapping between the source and target views. To test the usefulness of our filter on this type of applications, we run some experiments on a real-time 3D scanning application based on the Kinect. This application uses a structure from motion algorithm to gradually build the 3D geometry of the scene as we move around with the handheld Kinect [27]. The algorithm reconstructs the 3D geometry of the scene by aligning new frames on the already built model. Both filtered and the raw 3D data were tested (Fig. 19).

Registration based on the raw data was prone to misalignments which led to a rough 3D structure (particularly, when the object gets further away from the camera). As we can see in Fig. 19b, raw depth points lie in parallel planes (stripes can be clearly seen in column b). This shows what we have already explained in "3D image registration". The feature positions are discretised and distributed on the available depth levels. Although surface reconstruction algorithm fills the gaps between the planes after triangulation, the resulting model still suffers from a rough and bumpy surface. This drawback clearly appears after the lighting of the reconstructed structure. On the other hand, with the model resulting from the filtered data, illustrated in Fig. 19 column (c), the geometry is smoother and there is almost no misalignment between the views taken over time. The resulting 3D geometry is more realistic and less noisy. Hence, it does not need any further post-processing.

From the computational point of view, 3D reconstruction algorithm runs at 20 fps. This frame rate is less than the frequency of filtering which varies between 25 and 30 fps. No ground truth model was used to exactly evaluate the error after reconstruction (unlike what we did in the tracking application). Visually, we can assess the high quality of the outputs resulting from the scanning process (the filter transforms the points to their optimal 3D locations).

10 Conclusion and future work

In this work, we presented an innovative approach to enhance the quality of raw RGBD data issued by Kinect-like sensors. We built the mathematical model representing the depth map and successfully adapted it to the Kalman filtering scheme. To validate our finding, we tested the filtering approach on an object tracking algorithm (to quantify the accuracy of the data after applying the filter). In addition, we tested the output of the filter in a 3D scanning application (to visually assess its effect on the 3D model). We proposed a possible architecture to directly integrate the filter in the existing driver of the camera. However, to maintain the real-time nature of the sensor, we used a parallel algorithm that applies the same kernel of processing on all the pixels in the depth image. Practically, the filter was designed pixel-wise to avoid constraining any pixel by its neighbours. Thus, our own solution was implemented in the GPU.

As future work, we aim to extend our findings about the Kinect to other 3D scanning devices. This will allow us to reach a better accuracy without generating latency in the system. As a result, consumer cameras will be better endowed to reasonably achieve what could not be otherwise done without expensive sophisticated tools. We aim to fully embed the algorithm on dedicated computing boards to free the system from the hassle of filtering the raw 3D data. We furthermore initiated the application of the Kalman filter to RGBD image stabilisation. Most of the state-of-the-art approaches that are used come from image processing background. The benefit we got from this research can be similarly investigated in other image problems. The advantage of the Kalman filter is that it can produce optimal results with correct adaptation to the problem.

References

1. Brooks, F.P.: What's real about virtual reality? IEEE Comput. Graph. Appl. **19**(6), 16–27 (1999)
2. Kraus, K., Pfeifer, N.: Determination of terrain models in wooded areas with airborne laser scanner data. ISPRS J Photogramm Remote Sens **53**(4), 193–203 (1998)
3. Boehler, W., Vicent, M., Marbs, A.: Investigating laser scanner accuracy. Pediatr, Blood Cancer (2013)
4. W. Cady, Radar scanners and radomes. (2008)
5. Seitz, S.M., Curless, B., Diebel, J. Scharstein, D., Szeliski, R.: A comparison and evaluation of multi-view stereo reconstruction algorithms. In: Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on (CVPR'06), vol. 1, pp. 519–528 (2006)
6. de Aguiar, E., Stoll, C., Theobalt, C., Ahmed, N., Seidel, H.-P., Thrun, S.: Performance capture from sparse multi-view video. ACM Trans Graph **27**(3), 1 (2008)

7. Tong, J., Zhou, J., Liu, L., Pan, Z., Yan, H.: Scanning 3D full human bodies using Kinects. IEEE Trans Vis Comput Graph **18**(4), 643–650 (2012)
8. Kim, Y.M., Theobalt, C., Diebel, J., Kosecka, J., Miscusik, B., Thrun, S.: "Multi-view image and ToF sensor fusion for dense 3D reconstruction," In: Computer Vision Workshops, ICCV Workshops, 2009 IEEE 12th International Conference on, pp. 1542–1549 (2009)
9. Meinherz, C.: Time of flight camera unit and optical surveillance system. US Pat App 13/086,686 (2011)
10. Smisek, J., Jancosek, M., Pajdla, T.: "3D with Kinect." In: Computer Vision Workshops (ICCV Workshops). 2011 IEEE International Conference on, pp. 1154–1160 (2011)
11. Hall-Holt, O., Rusinkiewicz, S.: Stripe boundary codes for real-time structured-light range scanning of moving objects. In: Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001, vol. 2, pp. 359–366 (2001)
12. Menna, F., Remondino, F., Battisti, R., Nocerino, E.: Geometric investigation of a gaming active device. In: SPIE Optical Metrology, pp. 80850G-80850G. International Society for Optics and Photonics (2011)
13. Camplani, M., Mantecon, T., Salgado, L.: Depth-color fusion strategy for 3-D scene modeling with Kinect. IEEE Trans Cybern **43**(6), 1560–1571 (2013)
14. Kalman, R.E.: A new approach to linear filtering and prediction problems. J Basic Eng **82**(1), 35 (1960)
15. Ling, L., Cheng, E., Burnett, I.S.: An iterated extended Kalman filter for 3D mapping via Kinect camera. In: Acoustics, Speech and Signal Processing, 2013 IEEE International Conference on, pp. 1773–1777 (2013)
16. Hervier, T., Bonnabel, S., Goulette, F.: Accurate 3D maps from depth images and motion sensors via nonlinear Kalman filtering. In: Intelligent robots and systems, 2012 IEEE/RSJ International Conference on, pp. 5291–5297 (2012)
17. Park, S., Yu, S., Kim, J., Kim, S., Lee, S.: 3D hand tracking using Kalman filter in depth space. EURASIP J Adv Signal Process **2012**(1), 36 (2012)
18. Andersen, M., Jensen, T., Lisouski, P.: Kinect depth sensor evaluation for computer vision applications (2012)
19. Kalman, R.: A new approach to linear filtering and prediction problems. J. Basic Eng. **82**(no. Series D), 35–45 (1960)
20. Simon, D.: Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches. Wiley-Interscience, p. 552 (2006)
21. Khoshelham, K., Elberink, S.O.: Accuracy and resolution of Kinect depth data for indoor mapping applications. Sensors (Basel) **12**(2), 1437–1454 (2012)
22. Liebowitz, D., Zisserman, A.: Metric rectification for perspective images of planes. In: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231), pp. 482–488 (1998)
23. Fung, J., Mann, S.: Using graphics devices in reverse: GPU-based Image Processing and Computer Vision. IEEE Int. Conf. Multim. Expo **2008**, 9–12 (2008)
24. Jargstorff, F.: GPU image processing. SIGGRAPH (2004)
25. Kilgariff, E., Fernando, R.: The GeForce 6 series GPU architecture. ACM SIGGRAPH 2005 Courses (2005)
26. Arce, G.: Nonlinear signal processing: a statistical approach (2005)
27. Izadi, S., Davison, A., Fitzgibbon, A., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D.: KinectFusion. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology—UIST'11, p. 559 (2011)

Abdenour Amamra received his computer engineering degree from the Engineering College, Algeria, 2011. He has been a fellow researcher and PhD student at Cranfield University, UK since 2012.

Nabil Aouf received his PhD from the Department of Electrical Engineering, McGill University, Montreal, QC, Canada, in 2002. He has been a scientist for National Research Council Canada and also an Adjunct Assistant Professor with Concordia University, Canada.