Robotica (2016) volume 34, pp. 1855–1879. © Cambridge University Press 2014
doi:10.1017/S026357471400263X

Real-time multiview data fusion for object tracking with RGBD sensors

Abdenour Amamra* and Nabil Aouf

Centre For Electronic Warfare, Cranfield University, Defence Academy of the United Kingdom, Shrivenham, SN6 8LA.

(Accepted October 21, 2014. First published online: December 1, 2014)

SUMMARY
This paper presents a new approach to accurately track a moving vehicle with a multiview setup of red–green–blue depth (RGBD) cameras. We first propose a correction method to eliminate a shift, which occurs in depth sensors when they become worn. This issue could not be otherwise corrected with the ordinary calibration procedure. Next, we present a sensor-wise filtering system to correct for an unknown vehicle motion. A data fusion algorithm is then used to optimally merge the sensor-wise estimated trajectories. We implement most parts of our solution in the graphic processor. Hence, the whole system is able to operate at up to 25 frames per second with a configuration of five cameras. Test results show the accuracy we achieved and the robustness of our solution to overcome uncertainties in the measurements and the modelling.

KEYWORDS: RGBD; Real-time; Tracking; Multiview; Kalman filter; Robust H_∞; Covariance intersection; GPU.

1. Introduction

Nowadays, real-time tracking of moving objects in red–green–blue (RGB) video streams has become one of the most targeted research areas. Surveillance, sports reporting, video annotation, and traffic management systems are a few of the domains that have been widely benefited from advances in this field.[1] At the highest current performance level, the RGB data remains a limiting factor in giving a complete real world view. Recent off-the-shelf RGBD sensors, such as the Microsoft Kinect, provide great potential for the better perception of the surrounding space.[2] These cameras have the ability to deliver both 3D map of the scene and the corresponding colour image at a frequency of 30 Fps.

In the present work, we focus on the use of multiple RGBD sensors to accurately localise moving robots. The result of this solution is used to feed augmented reality and robotic systems with real-time pose data. The cooperative multiview sensing is better able to overcome the occlusions among different views. The latter leverages the joint action of all the sensors for more reliable tracking. Nevertheless, the processing of large amounts of 3D data is computationally expensive and as such may inversely affect the response time of the system. Hence, a need emerges for compromise between performance and response time in the processing stream. The graphic processing units (GPUs) are a very powerful tool when the different parts of the data can be simultaneously processed.[3] In our case, the huge amount of 3D data parallel streamed by the cameras is subject to smoothing and fusion algorithms. The processing starts with the RGBD data acquisition. The data is then sent through a smoothing stage to enhance its quality.[4] The 3D positions of the targets are then computed and sent to the robust H_∞ filtering level to correct them. Based on single estimates, a data fusion algorithm is applied to combine all the sensor-wise estimates of the position in a unique consistent result.

Our three main contributions in the present work are as follows:

- We improve the raw measurements of the RGBD sensor using a mathematical correction model.

* Corresponding author. E-mail: a.amamra@cranfield.ac.uk

- We deal with uncertainties in the model describing the motion of the vehicle using the robust H_∞ filter (RF), which has the ability to deal with uncertainties in the measurements and the modelling.
- We apply the Covariance Intersection (CI) algorithm with adaptive weighting coefficients computed from the specific characteristics of our tracking setup.

The paper is structured as follows: In Section 2, we discuss the state of the art of image-based tracking applications. We then explain the architecture of our system in Section 3. We give the details of the first two modules of the system in Section 4. In Section 5, we detail the modelling of uncertainties as to how the objects can be accurately tracked without prior knowledge of their motion. In Section 6, we present the CI technique. In Section 7, we clarify how it is possible to compute the orientation of the vehicle in our solution. We validate our finding in Section 8, where we plot error graphs obtained from the real experiments. Finally, we discuss the possible improvements and future works in Section 9.

2. Related Works

The tracking problem is generally divided into three stages: motion detection, object segmentation, and object tracking.[5] Single-camera tracking methods suffer from object/object or object/obstacle occlusions. The latter lead to failure as the tracked entities become incorrectly associated.[6] Lv *et al*.[7] presented a method for people tracking with a single camera. They used 3D shape models of people that were projected back into the image space to perform segmentation and resolve occlusions. Each human hypothesis is then tracked in 3D with a Kalman filter (KF) using the object's appearance constrained by its shape. Okuma *et al*.[8] propose a combination of Adaboost for object detection and particle filters for multiple objects' tracking. The combination of the two approaches leads to fewer failures than either one on its own, as well as addressing both detection and consistent track formation in the same framework. Li *et al.*[9] present a pedestrian detection algorithm for crowded scenes. His method iteratively aggregates local and global patterns for a better segmentation. These and other similar algorithms are challenged by the occluding and partially occluding objects and appearance changes.

On the other hand, the cooperative multiview object tracking has been recently benefited from a large interest against a single camera method.[6] This advantage is largely driven by a wider coverage of the scene, which is an asset in handling occlusions. The KidsRoom system developed at the MIT Media laboratory[10] uses a real-time tracking algorithm based on contextual information. The algorithm uses the overhead camera views of the space, which minimises the possibility of one object occluding another. The system can track and analyse the actions and interactions of people and objects. The lighting is assumed to remain constant during the runtime. The background subtraction is used to segment objects,[11] and the foreground pixels are clustered into 2D blobs. The algorithm then maps each person known to be in the room with a blob in the incoming image frame. Person-finder (Pfinder) is another real-time system for the tracking and interpretation of human motion.[12] Motion detection is performed using the background subtraction, where the statistics of background pixels are recursively updated using a simple adaptive filter. The human body is modelled as a connected set of blobs using a combination of spatial and colour cues. Pfinder has been applied in a variety of applications, including video games, distributed virtual reality, providing interfaces to information spaces, and recognising sign language. In our system, we only track robots in the first stage. We also apply background subtraction to extract the position of markers on the vehicle. However, the computation of the actual centre of mass is based on the extraction of contours for every marker and the computation of its corresponding zero-th and first moments.[13]

From a filtering point of view, the tracking problem is considered as a sequential recursive estimation problem. The estimation combines the knowledge about the previous estimate and the current measurement using a state/transition model. In the image space, the measurement comes from the relative 3D position of the vehicle in the space where it is moving. Each frame is processed within a time step. The noise statistics are deduced from the noise process affecting both the measured and the estimated positions. The state/space formalism, where the current tracked object properties are described in an unknown state vector updated by the noisy measurements, is very well adapted to the object tracking problem. The sequential estimation has an analytical solution under a very restrictive hypothesis. The KF is an optimal solution for the class of linear Gaussian estimation problems.[14]

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

For nonlinear systems, a number of Bayesian techniques were proposed to perform optimisation. When the Gaussian distribution is assumed, commonly used approaches include the Extended KF (EKF)[15] and the Unscented KF (UKF).[16] The particle filter is another numerical method that enables an approximate solution to the sequential estimation to be found.[17] All the above filters are very sensitive to the error in the system's model. In other words, if the system is imprecisely modelled, which is very common in real scenarios,[18] then the accuracy of the estimation is not optimal. To overcome uncertainty in the system, the robust H_∞ filter is known for its ability to cope with the uncertainties impinging on the model and the measurements. The authors are not aware of any RF system being used for accurate tracking. Furthermore, CI[19] is applied to combine the estimates computed upon the raw output of each camera in a way that minimises error in the global estimate.[20]

Red–green–blue depth sensors have recently become very popular in the capture of 3D data among professionals and researchers alike. These do not deliver just the colour but also the depth information at a frame rate of 30 Fps. This information (colour and depth) enables us to benefit from both the colour image and the 3D geometry approaches to overcome the traditional problems of many robotic and computer vision applications such as human pose estimation,[21] robot navigation,[22] SLAM,[23] object tracking,[24] and 3D scanning.[25] However, the 3D (x, y, z) point data resulting from these RGBD sensors are more important in size than their colour counterpart. Thus emerges the need for the GPUs to achieve real-time performance. There are many examples in published literature that show the bottlenecks in processing are reduced if the solution is implemented using the GPU, resulting in higher performance. Kinect fusion[25] and the work of Tong *et al*.[26] are the best examples.

3. System Overview

3.1. Kinect camera

Kinect sensor[1] is an RGBD camera that has the ability to capture a depth map of the scene and its RGB colour image at a frame rate of 30 Hz and a resolution of 640 × 480 pixels. The sensor contains an infrared (IR) projector that projects a set of IR patterns onto the scene. An IR camera captures the light reflected by the projected patterns and an RGB imager senses the colour of objects.[27] The sensor infers the depth of the scene after the computation of disparity with a triangulation approach. After a calibration procedure, both the RGB image and the depth data can be fused into a coloured point cloud of about 300,000 points in every frame.

Although the Kinect comes with factory-embedded calibration parameters (f_x, f_y, c_x, c_y intrinsic parameters for both RGB and IR cameras, and the extrinsic parameters $[R, T]$ for the IR-RGB stereo setup), the actual accuracy of the capture may range from less than 1 mm to many centimetres. This clear difference depends on the state of the sensor, the target application, and the nature of the scene.[28] To reliably track moving objects with Kinect, the sensor should be accurately recalibrated. The native parameters are more generic and remain the same for all the Kinects in the market. However, the frequency of use and the external factors, which widely differ from one application to another, can easily affect the precision of measurements.[24]

3.2. Hardware and software configuration

Our real-time tracking setup comprises $N = 5$ Kinect cameras covering a volume of 4 × 4 × 3 m. All the sensors are connected to the same workstation (Fig. 1). The hardware configuration encloses an INTEL i7 3930K CPU with six physical cores (two logical cores per physical core) running at 3.20 GHz, 16.0 GB of RAM as well as an NVIDIA GeForce 2-GB GTX 680 GPU. In the present research, we track a ground robot (Pioneer P3-DX).[2] However, our tracking system can be used to estimate the trajectory of any kind of ground or aerial vehicle moving in indoor environments. The robot moves freely in the space covered by the sensors according to an embedded obstacle avoidance algorithm that runs simultaneously with the capture. Therefore, we use a more general motion model that adapts to every possible model of motion characterising the vehicle. From a software point of view, we need to access the output of all Kinects simultaneously in real time. Thus, we used Kinect

[1] http://www.xbox.com/en-GB/Kinect (2012).
[2] http://www.mobilerobots.com/researchrobots/pioneerp3dx.aspx (2013).

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

Fig. 1. Multi-Kinects real-time tracking system.

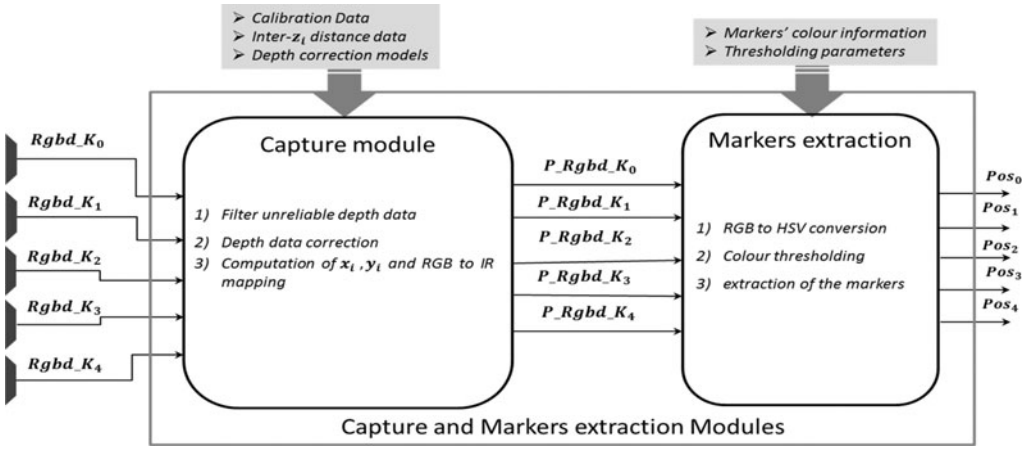


Fig. 2. Capture and markers extraction modules.

SDK 1.7.0[3] and CUDA[4] to program the GPU. CUDA is a GPU/CPU programming language that allows us to build heterogeneous applications capable of running in the CPU and some kernels to be launched in the GPU.[29]

3.3. Real-time multi-Kinects tracking architecture

The system comprises the following four main modules through which flows every synchronised set of RGBD data streamed by five sensors (see Figs. 2 and 3):

1. Capture module

   It streams 3D point clouds to the tracker and to the following stages of the platform. At this level, we associate a thread to each sensor. This architecture allows full occupation of both CPU and GPU during the capture. Nevertheless, other sensor-related limitations should be taken into account when using multiple Kinects simultaneously. The IR beams emanating from the projectors can interfere with each other. They confuse IR cameras when inferring the disparity as it would be impossible to decide which IR speckle belongs to which sensor. As a result, some holes may appear in 3D data because of the undefined disparity information.[30] Each thread acts independently by loading the data into the GPU and performing the following computations:

[3] http://www.microsoft.com/en-gb/download/details.aspx?id=36998 (2013).
[4] http://www.nvidia.com/object/cuda_home_new.html (2013).

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

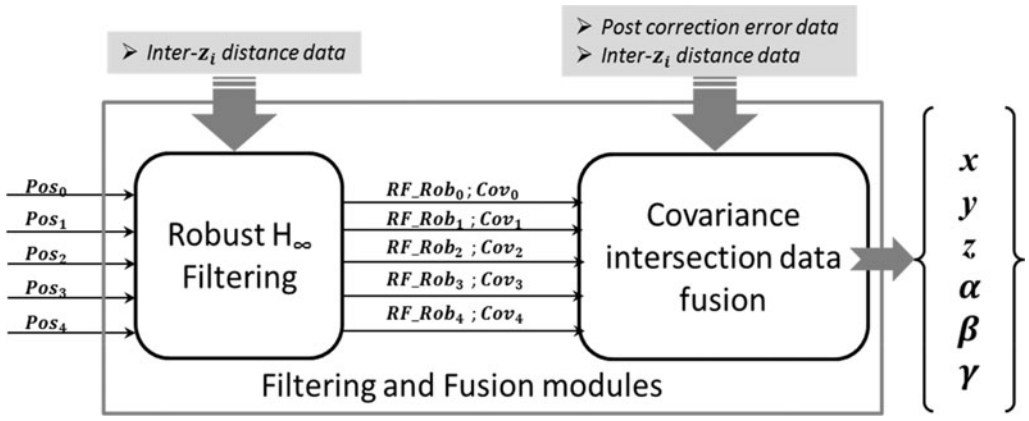


Fig. 3. Filtering modules.

- Filtering the unreliable z_i elements (pixels where no disparity information is available).
- Correcting the remaining valid depth values using appropriate correction models.
- Computing x_i, y_i for the valid points only using the intrinsic parameters of IR camera.
- Mapping the colour image onto the depth image using stereo calibration parameters.

2. Markers extraction module

To compute position and orientation of robot, we fix three distinctive markers on its top (see Fig. 1). The 3D pose of a moving object is obtained by estimating its centre of mass and the corresponding orientation. This becomes possible because we have the colour data mapped with the depth image. Consequently, we just need to fetch the markers in the colour space, and then the corresponding 3D positions are easily resolved. The background/foreground segmentation module takes as input the aligned colour image and follows the following steps:

- RGB to Hue, Saturation, and Value (HSV) conversion. This conversion is motivated by the fact that the HSV space is more robust to light intensity changes.[31]
- Colour thresholding to separate markers from background.
- Erode and dilate binary thresholded image.
- Actual extraction of markers.

The output of this module is the raw position and orientation data characterising the vehicle in its neighbourhood.

3. The robust filtering module

The filtering module aims to enhance the quality of position and the orientation information. It acts on the whole trajectory of a moving robot, and filters it according to a rough state/transition motion model. However, for generality and to allow the solution to work with any kind of ground or aerial vehicles, we assume that we do not have an exact model. We choose a classical Newtonian motion system with approximate parameters. We compensate for the lack of knowledge about the nature of the system by adapting the robust H_∞ filtering scheme[32] to our Newtonian model of motion. The output of this stage is the sensor-wise filtered positions and orientations.

4. Data fusion module

At this level, we aim to combine all the sensor-wise estimates to produce, in a cooperative manner, a complete and more accurate position data. We use the external calibration parameters of Kinects with the reference frame to combine the data with the CI technique[19,33] by following these steps:

- Transforming each position data for every Kinect to a global reference frame.
- Applying the CI algorithm on these data to produce a unique acceptable position and orientation information.

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

As the markers can be occluded during tracking, it is necessary to fill the missing data. In addition, the quality of measurements is not the same among the five viewpoints. Thus, a weighting coefficient is associated to each sensor according to the error in its measurements. These coefficients promote more accurate measurement. This approach allows us to optimally fuse all the outputs in a more precise estimate benefitting from the best of each sensor.

4. Capture and Markers Extraction

4.1. Capture module

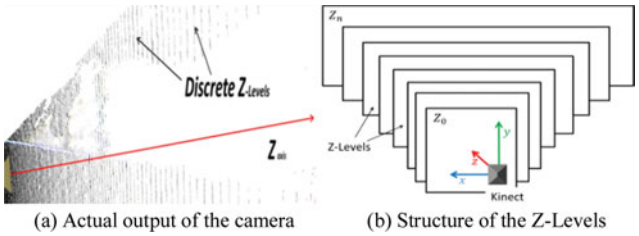
1. Filtering the unreliable z_i
   We filter the unreliable z_i to reduce the markers' search space in the corresponding RGB image. A depth value is considered unreliable if:
   - there is no disparity information at the raw depth pixel, and
   - it exceeds the interval of useful depth data (0.8–4.5 m).
2. z_i correction with sensor's model
   After being used for a long time, every electronic device suffers from a decrease in accuracy. The quality of the 3D map obtained by the Kinect sensor is highly affected by the performance of its IR setup. However, the RGB camera (passive part of the device) is more robust and the colour image remains unaffected for a longer time.
   To ensure a higher precision in our tracking system, we propose an effective correction approach to enhance the quality of measurements among different Kinects covering the same scene. To this end, we compute a function that takes as input the shifted raw outputs of device and corrects them. This approach is justified by the fact that the regular camera calibration procedure cannot handle this drop in the quality of disparity measurements. The origin of the problem is not related to the optical configuration of IR camera but to an error factor generated by the disparity computation module.[34] The latter is an offset that appears in the estimation of distance between the sensor and the object. Its value becomes more important as the object gets further away. Our purpose is to correctly remap the shifted depth (z_{sh}) to its respective correct value. Hence, we take the depth measurements output by the sensor along with their correct ground truth counterparts (z_{cor}), the pairs (z_{sh}, z_{cor}) are related by a function

$$f(z_{\text{sh}}) = z_{\text{sh}} - z_{\text{cor}}, \tag{1}$$

   where $f(z_{\text{sh}})$ represents the shift from the correct reading. It computes for every depth value the corresponding error based on the ground truth reference (z_{cor}) as shown in Eq. (1). $f(z_{\text{sh}})$ is computed from the sample points taken simultaneously from Kinect and a high precision tracking system.[5] Once we obtain the pair (z_{sh}, z_{cor}), we fit the sparse data with a polynomial approximating the shape of representative curve. The same polynomial will serve as a model, which takes as input a raw measurement (z_{sh}) generated by the Kinect and outputs a corrected estimate (z_{cor}). Nonetheless, the shift is not calculated for all the 300,000 points in the frame. Instead, we focus on correcting a limited set of known Z-levels (Fig. 4) that can exist in the point cloud.[35] Thus, we attribute to each raw range value a respective corrected value. As a result, the correction of the whole depth image is reduced to the correction of the known Z-levels only.
3. x_i, y_i computation and stereo mapping
   Kinect has two cameras, one for the colour image and the other for the depth data. However, we do not actually know the depth of a given pixel in the colour image because the two cameras have different viewpoints. Hence, we need a proper stereo calibration to get the right [**R**, **T**] transformation that relates both cameras. We first compute the world coordinates x_i, y_i or every valid pixel coordinates (Eqs. (2) and (3)). To complete this step, we need the calibration parameters

[5] http://www.naturalpoint.com/optitrack/ (2013).

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press



(a) Actual output of the camera (b) Structure of the Z-Levels

Fig. 4. Kinect depth data. (a) Actual output of camera. (b) Structure of Z-levels.

of the IR camera:

$$x_i = (u_{i\_ir} - c_{x\_ir})\ z_i / f_{x\_ir}, \tag{2}$$

$$y_i = \left(v_{i\_\mathrm{ir}} - c_{y\_\mathrm{ir}}\right)\ z_i / f_{y\_\mathrm{ir}}. \tag{3}$$

Afterwards, we apply the stereo calibration parameters [**R**, **T**] on the point $P\ (x_i, y_i, z_i)$ to transform it from the IR coordinate system to the RGB frame $P'\left(x_i', y_i', z_i'\right)$, Eq. (4):

$$P' = RP + T. \tag{4}$$

In the next step, we re-project P' to RGB imager using the intrinsic parameters of colour camera (Eqs. (5) and (6)),

$$u_{i\_\mathrm{rgb}} = \left(x_i'\ f_{x\_\mathrm{rgb}}/z_i'\right) + c_{x\_\mathrm{rgb}}, \tag{5}$$

$$v_{i\_\mathrm{rgb}} = \left(y_i'\ f_{y\_\mathrm{rgb}}/z_i'\right) + c_{y\_\mathrm{rgb}}. \tag{6}$$

The output is a coloured 3D point cloud where every point $P(x_i, y_i, z_i)$ has its own colour and world coordinates.

4.2. Markers extraction

1. RGB to HSV conversion
   The RGB colour model is often used in computer and electronic systems. However, this coding has many downsides when we consider colour from a perceptual point of view.[31] When we try to decide whether an object of a known colour exists in a given image like a human does, we note a large difference between the respective RGB combinations of source and target objects. This even happens when there occurs a small change in the lighting. On the other hand, the HSV colour coding was proven to be less sensitive to light intensity and shadows.[36]
2. Colour thresholding and morphological operations
   Once we convert our image to HSV space (Fig. 5(a)), we need to localise the areas of similar colour as the target. The thresholding and binarisation (Fig. 5) are used to recognise and extract three yellow markers from the scene. The alignment between the colour image and the depth map allows us to obtain 3D coordinates for each marker. We apply erosion on binary image to eliminate disturbing noise spots, followed by a dilation to recover the eroded part of the areas representing the markers. The 3D position of the robot is the centre of the triangle defined by the three markers. On the other hand, the orientation of the robot is deduced from the centre of the triangle and the frontal marker.
3. Markers extraction

After localising the markers in the binary image, we proceed to the actual computation of their centre of mass. We start by extracting the contours of every marker in the binary image (white spots in Fig. 5(b)). Afterwards, we compute the zero-th and first moments of each marker in the binary

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

(a) HSV image (b) binary image (c) tracked markers

Fig. 5. Markers extraction. (a) HSV image. (b) Binary image. (c) Tracked markers.

image[13] using Eq. (7):

$$\mu_{m,n} = \sum_{x=0}^{\infty} \sum_{y=0}^{\infty} x^m y^n f(x, y). \tag{7}$$

To compute the centroid of the markers in our binary image, we use the first moments μ_{01}, μ_{10} and the moment μ_{00} (represents the area covered by the marker). The marker's centroid coordinates are

$$(x_0, y_0) = \left(\frac{\mu_{10}}{\mu_{00}}, \frac{\mu_{01}}{\mu_{00}} \right). \tag{8}$$

This technique is robust to noise. The centroid might have little bit shifted because of some noisy contour elements. However, the error in its position does not significantly affect the accuracy of our tracker even if the target is further away.

5. Robust Filtering

5.1. Motion model

Raw Kinect position measurements are only precise at a close range from the sensor because the error in position remains below 5 cm for the 3-m correction module we introduced. It becomes below 5 cm for up to 4.5-m depth after the correction. However, the accurate tracking of moving objects requires a higher accuracy with an acceptable additional computational load. To fulfil this requirement in a relatively large indoor space, we apply the robust H_∞ filtering[32] on the trajectories delivered by the five cameras. Our need for such a filtering scheme is motivated by its ability to deal with the problem of uncertainty in the model describing the motion of the vehicle. If the filter becomes too tight to the imprecise model of the vehicle, the tracking would fail within a few iterations. In the context of this work, the tracked entities are assumed to move irregularly. The translation, the rotation, and the occlusions between the rigid bodies often occur in real situations. As a consequence, a constant velocity model is difficult to adapt. On the other hand, the acceleration of ordinary ground and flying robots is more stable and does not change in magnitude just as it does in sign because of smooth and gradual variations in velocity. Consequently, the motion of the vehicle tends to follow a Newtonian model with a varying velocity and a bounded acceleration. To correct the raw position data output by the cameras in real time, the filter should be able to robustly predict the next state of the vehicle $[x_k y_k z_k \dot{x}_k \dot{y}_k \dot{z}_k]^T$ and correct it accordingly after obtaining the measurements and the control input (acceleration). However, the filter is not applied to the orientation data. The computation of the latter is based on the estimated positions of the markers and the centroid of the robot.

For the (x, y, z) position of a given marker, the motion model will be

$$\begin{cases} x_{k+1} = x_k + T\dot{x}_k + \frac{T^2}{2}\ddot{x}_k, \\ y_{k+1} = y_k + T\dot{y}_k + \frac{T^2}{2}\ddot{y}_k, \\ z_{k+1} = z_k + T\dot{z}_k + \frac{T^2}{2}\ddot{z}_k. \end{cases} \tag{9}$$

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

The equations of the corresponding velocities $(\dot{x}, \dot{y}, \dot{z})$ are:

$$\begin{cases} \dot{x}_{k+1} = \dot{x}_k + T\ddot{x}_k, \\ \dot{y}_{k+1} = \dot{y}_k + T\ddot{y}_k, \\ \dot{z}_{k+1} = \dot{z}_k + T\ddot{z}_k. \end{cases} \tag{10}$$

The state-transition system is

$$\begin{aligned} s_{k+1} &= Fs_k + Bu_k + w_k, \\ t_k &= Hs_k + v_k, \end{aligned} \tag{11}$$

where:

$$\begin{aligned} s_k &= [x_k y_k z_k \dot{x}_k \dot{y}_k \dot{z}_k]^T, \\ t_k &= \left[\breve{x}_k \breve{y}_k \breve{z}_k\right], \\ u_k &= [\ddot{x}_k \ddot{y}_k \ddot{z}_k], \\ H &= [I_3, 0_3]. \end{aligned} \tag{12}$$

At every time-step k,s_k is the estimated state of the vehicle (position and velocity); t_k is the measurement output by the sensor; u_k is the acceleration of the vehicle along the three axes; Q_k is the covariance of noise affecting the system (W_k); and R_k is the covariance of noise affecting the measurements (v_k).

From Eqs. (9) and (10), the state-transition matrix F becomes Eq. (13),

$$F = \begin{bmatrix} I_3 & TI_3 \\ 0_{3,3} & I_3 \end{bmatrix}, \tag{13}$$

$$B = \begin{bmatrix} \frac{T^2}{2} I_3 \\ TI_3 \end{bmatrix}. \tag{14}$$

5.2. Robust H_∞ filter

In practical situations, the exact model of the system may not be available. The performance of such a system becomes an important issue. The robust H_∞ filter,[18] adopts as process and measurement models the state space representation in Eq. (15). In our case, the matrices of such models are defined by Eqs. (13) and (14):

$$\begin{aligned} s_{k+1} &= (F_k + \Delta F_k)\, s_k + Bu_k + w_k, \\ t_k &= (H_k + \Delta H_k) s_k + v_k. \end{aligned} \tag{15}$$

At time-step k, w_k and v_k are uncorrelated zero-mean white noise processes with the covariance matrices Q_k and R_k respectively. The matrices ΔF_k and ΔH_k represent the uncertainties in the system and the measurement matrices. These uncertainties are assumed to be of the form

$$\begin{bmatrix} \Delta F_k \\ \Delta H_k \end{bmatrix} = \begin{bmatrix} M_{1k} \\ M_{2k} \end{bmatrix} \Gamma_k N_k, \tag{16}$$

where M_{1k}, M_{2k}, and N_k are three known matrices, and Γ_k is an unknown matrix satisfying the bound:

$$\Gamma_k^T \Gamma_k \leq I. \tag{17}$$

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

It is assumed that F_k is non-singular. This assumption is not too restrictive; F_k should also be non-singular for real systems because it comes from the exponential of the system matrix (the matrix exponential is always non-singular). The problem is to design a state estimator of the form:

$$s_{k+1} = \tilde{F}_k s_k + K_k t_k \tag{18}$$

with the following characteristics:

- The estimator should be stable (the eigenvalues of $\tilde{F}_k$ should be less than 1 in magnitude).
- The estimator error satisfies the following worst-case bound:

$$\max_{w_k, v_k} \frac{\tilde{s}_{k2}}{w_{k2} + v_{k2} + \tilde{s}_{0_{o_1^{-1}}} + s_{0_{o_2^{-1}}}} \leq \frac{1}{\theta}. \tag{19}$$

- The estimation error of $\tilde{s}_k$ satisfies the following root mean square (RMS) bound:

$$E\left(\tilde{s}_k \tilde{s}_k^T\right) < P_k. \tag{20}$$

The solution of this problem can be achieved by the following procedure:

1. Choose a scalar sequence $\alpha_k > 0$ and a small $\varepsilon > 0$.
2. Define the following matrices

$$\begin{aligned} R_{11k} &= Q_k + \alpha_k M_{1k} M_{1k}^T, \\ R_{12k} &= \alpha_k M_{1k} M_{2k}^T, \\ R_{22k} &= R_k + \alpha_k M_{2k} M_{2k}^T. \end{aligned} \tag{21}$$

3. Initialise P_k and $\tilde{P}_k$ as follows:

$$\begin{aligned} P_0 &= O_1, \\ \tilde{P}_0 &= O_2, \end{aligned} \tag{22}$$

   where O_1, O_2 are the initial values that we attribute to the estimation error covariance matrices for the computation of R_{1k}, R_{2k}, F_{1k}, H_{1k}, and T_k. Although these parameters have initially large values, the filter automatically tunes them within few iterations. Hence, the process reaches a steady state, and the error in estimation decreases to its lowest levels.
4. Find the positive definite solutions P_k and $\tilde{P}_k$ satisfying the following Riccati equations:

$$\begin{aligned} P_{k+1} = {} & F_{1k} G_k F_{1k}^T + R_{11k} + R_{11k} R_{2k} R_{11k}^T \\ & - \left(F_{1k} G_k H_{1k}^T + R_{11k} R_{2k} R_{12k}\right) R_k^{-1} \left(F_{1k} T_k H_{1k}^T + R_{11k} R_{2k} R_{12k}\right)^T + \varepsilon I, \end{aligned} \tag{23}$$

$$\tilde{P}_{k+1} = F_k \tilde{P}_k F_k^T + F_k \tilde{P}_k N_k^T (\alpha_k I - N_k \tilde{P}_k N_k^T)^{-1} N_k \tilde{P}_k F_k^T + R_{11k} + \varepsilon I, \tag{24}$$

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

where the matrices R_{1k},R_{2k},F_{1k}, H_{1k}, and G_k are defined as:

$$R_{1k} = \left(\tilde{P}_k^{-1} - N_k^T N_k/\alpha_k\right)^{-1} F_k^T, \tag{25}$$

$$R_{2k} = R_{1k}^{-1}\left(\tilde{P}_k^{-1} - N_k^T N_k/\alpha_k\right)^{-1} R_{1k}^{-T}, \tag{26}$$

$$F_{1k} = F_k + R_{11k}R_{1k}^{-1}, \tag{27}$$

$$H_{1k} = H_k + R_{12k}^T R_{1k}^{-1}, \tag{28}$$

$$G_k = (P_k^{-1} - \theta^2 I)^{-1}. \tag{29}$$

5. If the Ricatti equation solutions satisfy:

$$\frac{1}{\theta^2} I > P_k, \tag{30}$$

$$\alpha_k I > N_k \tilde{P}_k N_k^T, \tag{31}$$

then the estimator of Eq. (18) solves the problem with:

$$K_k = \left(F_{1k}T_k H_{1k}^T + R_{11k}R_{2k}R_{12k}\right)\tilde{R}_k^{-1}, \tag{32}$$

$$\tilde{R}_k = H_{1k}T_k H_{1k}^T + R_{12k}^T R_{2k} R_{12k} + R_{22k}, \tag{33}$$

$$\hat{F}_k = F_{1k} - K_k H_{1k}. \tag{34}$$

The parameter ε is generally chosen as a very small positive number. In our case it was fixed to $\varepsilon = 10^{-8}$.

The parameter α_k has to be chosen large enough so that the conditions of Eqs. (30) and (31) are satisfied. However, when α_k increases, P_k also increases, which results in a looser bound for the RMS estimation error.[18]

A steady state of robust filter can be obtained by letting the parameter $P_{k+1} = P_k$ and $\tilde{P}_{k+1} = \tilde{P}_k$ in Eq. (24). In our case, we compared the raw tracking results delivered by the Kalman filter against the trajectory filtered with the robust H_∞ filter. With the application of this filter we have a more robust tracker.

The adaptation of the robust H_∞ filtering scheme is proven to be flexible and able to produce accurate state estimation based on uncertain system parameters. This asset enables us to track vehicles without the exact knowledge of their motion model. The robust H_∞ filter showed very interesting results in several automation and control applications.[37,38] More importantly, the results we obtained and the error in estimation compared with the ground truth measurements show the effictivness of our approach against the naïve filtering scheme (KF does not consider uncertainties in the system). The latter does not have the ability to cope with uncertainty. Nevertheless, if the exact model becomes available, the robust H_∞ filter performs even better. Although in many real cases the exact model is hard to determine.[39] The robust filter combines the robustness of H_∞ (it is less affected by the accuracy of system's parameters) and the optimality of Kalman filtering.

6. Tracking Data Fusion

The precision of some Kinects' tracking measurements can be important, particularly if the target moves far from them. In addition, if there are many targets moving around the obstacles in the same scene, occlusions can appear and consequently prohibit the decent recognition of vehicles. At this level of our pipeline, we propose to develop cooperation between multiple Kinects with the application of CI filter.[19] We combine the estimated positions delivered by all the cameras (after being filtered by robust H_∞ filter) in one consistent estimate that precisely determines the pose of the vehicle in its space.

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

6.1. Covariance intersection filtering

Based on the estimates determined by the robust H_∞ filter $\hat{x}_{kn}$ $(0 \leq n \leq N)$ at time step k, and their respective covariance matrices P_{kn}, we compute a combined estimate $\tilde{x}$ with its error covariance P where the true state of the system is x (real position of the vehicle).

If we consider N unbiased estimates related to each camera $\hat{x}_1, \hat{x}_2, \hat{x}_3 \ldots \hat{x}_N$ for the unknown state vector $\tilde{x}$:

$$\tilde{x} = P \sum_{n=1}^{N} P_n^{-1} \hat{x}_n, \tag{35}$$

$$P^{-1} = \sum_{n=1}^{N} P_n^{-1}. \tag{36}$$

In the presence of a correlation between estimation errors, the estimated P may become far too optimistic and this can cause divergence in sequential filtering. A conservative estimate can be given by applying CI according to:

$$\tilde{x} = P \sum_{n=1}^{N} \omega_n P_n^{-1} \hat{x}_n, \tag{37}$$

$$P^{-1} = \sum_{n=1}^{N} \omega_n P_n^{-1}, \tag{38}$$

with the non-negative coefficient ω_n verifying the following consistency condition:

$$\sum_{n=1}^{N} \omega_n = 1. \tag{39}$$

An estimate can always be obtained with

$$P \geq P_0 := E\left[(\tilde{x} - x)(\tilde{x} - x)^T\right], \tag{40}$$

where $P \geq P_0$ denotes the fact that $P - P_0$ is positive semi-definite. Consequently, the coefficient ω_n is meant to minimise either the trace or the determinant of P. In order to avoid the possibly high numerical implementation effort to find the solution of this nonlinear optimisation problem, Niehsen[19] has proposed a fast approximate solution.

For $trace\,(P_n) \leq trace\,(P_m)\,; 1 \leq n,\ m \leq N$, one would expect $\omega_n \geq \omega_m$.

From a computationally optimal point of view, rather than using the estimation uncertainty P_n, the authors in ref. [33] introduced estimation certainty by considering $S_n = P_n^{-1}$:

$$\omega_n = \frac{\text{trace}(S_n)}{\sum_{i=0}^{N} \text{trace}(S_i)}. \tag{41}$$

Equation (41) means that the greater the trace(S_n) (the more certain we are about the estimate $\hat{x}_n$), the higher the corresponding weight ω_n. On the other hand, the smaller the trace(S_n), the lower the weight ω_n. More importantly, the consistency condition (39) remains satisfied:

$$\sum_{n=1}^{N} \omega_n = \frac{\sum_{n=0}^{N} \text{trace}(S_n)}{\sum_{i=0}^{N} \text{trace}(S_i)} = 1 \tag{42}$$

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

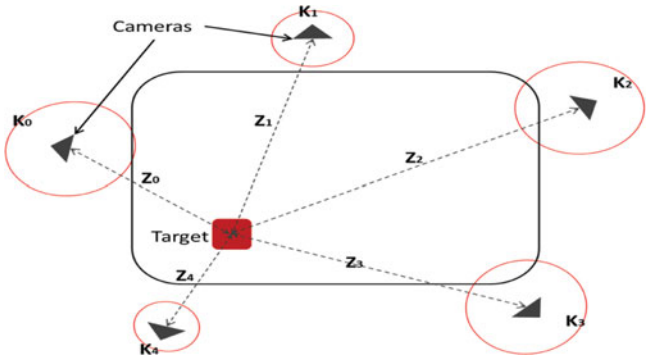


Fig. 6. CI parameters.

6.2. *Covariance intersection for multi-Kinect tracker*

As we have explained earlier, some Kinects may be faulty and consequently produce erroneous measurements when the target is far away from the sensor. However, with a cooperative multiview setup, the final estimate of position and orientation can be jointly corrected. The correction is led by the weighting coefficients which give higher weighting to the more accurate measurements delivered by each camera in the multiview setup covering the whole scene.

Another contribution of the present work is the adaptive weighting scheme based on the assessment of the quality for each estimate resulting from the robust H_∞ filter and the confidence in the raw measurements delivered by the camera itself. Indeed, we introduce a quality factor for each of the cameras capturing the motion of the vehicle. This quality indicator is obtained from the remaining error in the sensor after applying appropriate correction model. The mathematical formulation is as follows.

For the processing thread of the nth camera:

- P_n is the covariance matrix of the error in the estimate delivered by the robust H_∞ filter.
- K_n is the covariance matrix characterising the residual error after correction.
- Z_n is a positive scalar factor representing the distance between the target and the camera.

The last assumption is motivated by the fact that the smaller the depth of the target, the more accurate the resulting measurement.

Figure 6 depicts the situation where every sensor has its native hardware accuracy matrix K_n (red circles). Moreover, using the distance separating the camera from the target, we introduce the weighting coefficient Z_n.

For every pair of position estimates $\hat{x}_n, \hat{x}_m$, the following condition should be satisfied:

$$\begin{aligned} &\operatorname{tr}(K_n) + \operatorname{tr}(P_n) + Z_n \leq \operatorname{tr}(K_m) + \operatorname{tr}(P_m) + Z_m \Rightarrow \omega_n \geq \omega_m; \\ &1 \leq n,\ m \leq N. \end{aligned} \tag{43}$$

Equation (43) means that $\hat{x}_n$ affects the final estimate $\tilde{x}$ more than $\hat{x}_m$ does and P_n affects the final error in estimation P more than P_m does. In our tracking algorithm, we considered Niehsen's[19] findings about fast CI. In addition, we included the uncertainty characterising the quality of the measurements delivered by the sensor. Our weighting coefficients are given by the following expression:

$$\omega_n = \frac{\sum_{\substack{i=1 \\ i \neq n}}^{N} (\operatorname{tr}(K_i) + \operatorname{tr}(P_i) + Z_i)}{\sum_{i=1}^{N} (\operatorname{tr}(K_i) + \operatorname{tr}(P_i) + Z_i)}. \tag{44}$$

Another form of the same expression is more adequate to reduce the load of computation:

$$\omega_n = \frac{\sum_{i=1}^{N} (\operatorname{tr}(K_i) + \operatorname{tr}(P_i) + Z_i) - (\operatorname{tr}(K_n) + \operatorname{tr}(P_n) + Z_n)}{\sum_{i=1}^{N} (\operatorname{tr}(K_i) + \operatorname{tr}(P_i) + Z_i)}. \tag{45}$$

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

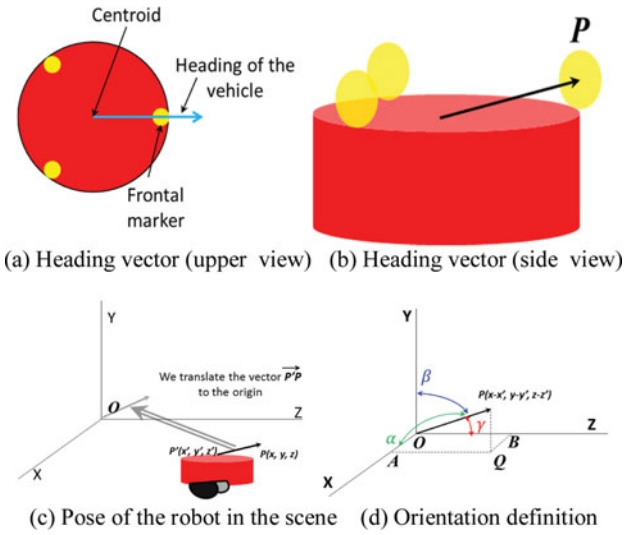


Fig. 7. Orientation computation. (a) Heading vector (upper view). (b) Heading vector (side view). (c) Pose of the robot in the scene. (d) Orientation definition.

Consequently, we only need to compute the traces of the matrices once. The denominator $\sum_{i=1}^{N}(\mathrm{tr}(K_i)+\mathrm{tr}(P_i)+Z_i)$ is also computed once. Afterwards we subtract the corresponding parameter $(\mathrm{tr}(K_n)+\mathrm{tr}(P_n)+Z_n)$ appropriate to every one of the estimates. The condition of consistency (39) remains verified as $\sum_{n=1}^{N}\omega_n = 1$. The experiments conducted using this approach proved that Eq. (45) is more realistic and suitable for real tracking scenarios.

7. Orientation Computation

Until this point, we have not discussed the computation of the direction towards which the robot is heading. However, our current solution is already able to deliver 6 DoF without any need to process orientation data independently. In the published literature, the common way to compute orientation during the motion of the vehicle is the well-known slow least square-based algorithms. This category of solutions produces elementary rotations and translations between the successive states taken by the robot. Thus, we considered it to be unsuitable for our real-time solution.

Once we obtain an accurate estimate of centroid, we can easily compute the remaining three orientation components (α, β, γ) by applying some simple trigonometry on the accurate positions of the centroid and the frontal marker (Figs. 7(a) and (b)). This method is effective as it prevents us complicating the filter with the extra load of computations regarding the orientation,

$$\mathrm{Mag} = \sqrt{(x-x')^2+(y-y')^2+(z-z')^2}, \tag{46}$$

$$\alpha = \arccos\left(\frac{x-x'}{\mathrm{Mag}}\right), \tag{47}$$

$$\beta = \arccos\left(\frac{y-y'}{\mathrm{Mag}}\right), \tag{48}$$

$$\gamma = \arccos\left(\frac{z-z'}{\mathrm{Mag}}\right). \tag{49}$$

Figures 7(c) and (d) illustrate how the problem of defining the orientations can be formulated. From Eqs. (47)–(49), one can directly obtain the correct angles that define where the vehicle is heading without any confusion. In addition, it is possible to compute the 3D rotation between two different orientations by just using the angles characterising the two poses.

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

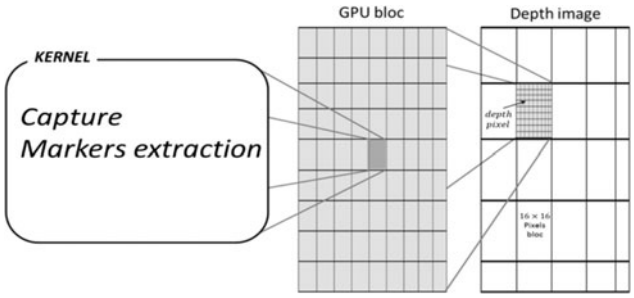


Fig. 8. KF GPU implementation for depth map filtering.

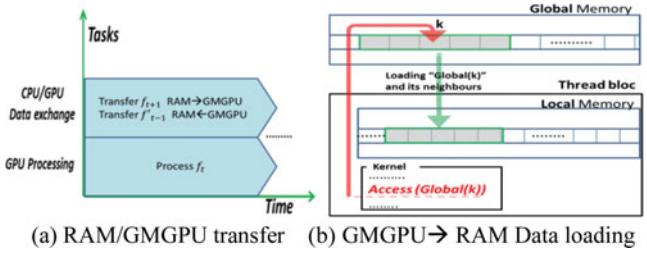


Fig. 9. Optimisation of data exchange in the GPU. (a) RAM/GMGPU transfer. (b) GMGPU → RAM data loading.

8. Results and Discussion

All the following stages are based on our hardware configuration. For programming, we used C++ and CUDA for the GPU/CPU heterogeneous coding.

8.1. GPU implementation of the capture and marker extraction algorithms

The subject images we are processing have a VGA resolution (640 × 480) delivered at the same frame rate of the camera to allow the following applications to fully exploit the frame rate offered by the sensor. When we first ran the correction on a regular CPU, the maximum achieved frame rate was 15 fps. Hence, emerges the need to implement the bottlenecks of our solution in the GPU. On the other hand, the image data is more naturally organised to fit GPU blocks, where every element in the block (thread) processes a single pixel at time.[40] Figure 8 illustrates how the depth image output by the camera is divided into image blocks of a constant size (16 × 16 pixels; so 256 threads is the size of the block in our implementation). The pixels of the same image block are processed simultaneously in the same GPU thread block. As a result, for every pixel in the image there is an attributed thread in the GPU. The latter runs the actual kernel on a single depth pixel (range reading). 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 computational ability of the GPUs. The complexity of processing is reduced to the complexity of the algorithm running in the Kernel, which indeed is constant.

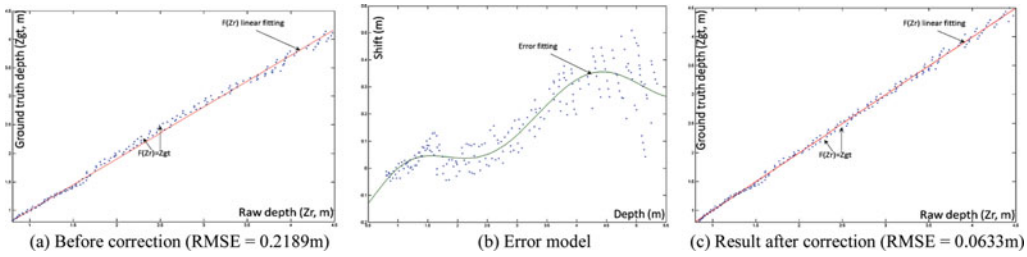
Other considerations should be addressed to optimise the utilisation of all the available hardware capability. Basically, the design of heterogeneous algorithms aims at a higher occupancy of processors and the full usage of the bandwidth when exchanging data between the central memory (RAM) and the global memory of GPU (GMGPU).[41] To this end, we focus on the following two optimisation aspects:

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 to exchange 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. 9(a)).

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 requested sooner as well.[42] Memory coalescing is another optimisation that significantly helps to

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press



(a) Before correction (RMSE = 0.2189m) (b) Error model (c) Result after correction (RMSE = 0.0633m)

Fig. 10. Drifty Kinect. (a) Before correction (RMSE = 0.2189 m). (b) Error model. (c) Result after correction (RMSE = 0.0633 m).

increase the probability of threads in the same warp (a group of 32 threads from the same thread block running simultaneously) to fetch the data from the memory together. The purpose of memory coalescing is to ensure that the threads access 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 happen automatically. Programmatically, structure of arrays rather than the easy use of array of structures 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 9(b) illustrates what happens when a thread fetches a given cell in the global memory.

The CPU and the GPU are significantly different. A GPU can handle large amounts of data in many streams, performing relatively simple operations, but it is inadequate for heavy processing on a single or few streams. A CPU is much faster on a per-core basis and can perform complex operations on a single or few streams of data more easily. Consequently, the robust filter and the CI algorithms have not been implemented in the GPU. The reason is that they do not interact with a large amount of image data. Hence, they just smooth the 3D positions of markers. More importantly, we were able to filter the five position data using a CPU-based multithreaded architecture where each thread handles the stream of a given camera. The fusion algorithm is then executed on the resulting estimates to find the correct pose of the vehicle.

8.2. Sensor first stage correction

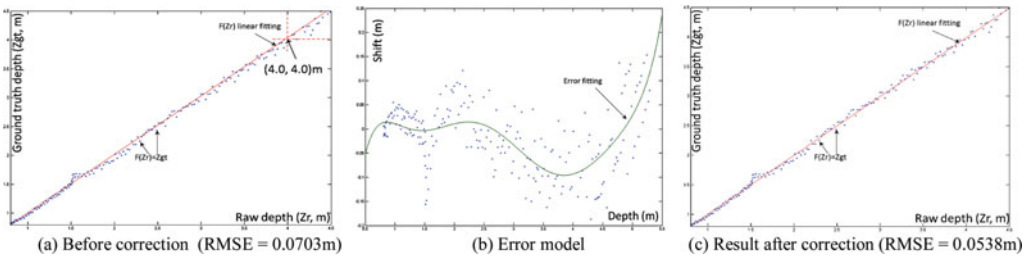
To illustrate the effect of our sensor correction method, we conducted some experiments where two Kinects of different measurement precisions were corrected. Figures 10 and 11 illustrate two cases where drifty and healthy sensors were adjusted using an 8° polynomial. The first column of both figures (column (a) in Figs. 10 and 11 before correction) presents the graphs of the function $g\,(z_{\text{sh}}) = z_{\text{cor}}$ (z_{sh} : shifted depth; z_{cor}: ground truth depth). The more the fitting line approaches $y = x$, the more accurate is the sensor (the range measurement delivered by the Kinect is closer to the ground truth). With faulty sensor (Fig. 10), the curve representing $g\,(z_{\text{sh}})$ is clearly shifted below $y = x$. This happens because the sensor overestimates the range of objects in front of it. As a consequence, such behaviour limits the ability of the sensor to fully capture the 3D geometry within the working range (0.8 to 4.0 m). On the other hand, with a good sensor (Fig. 11), the representative curve is almost superimposed on $y = x$ when the depth of the object is below 4.0 m. When the object goes further than permitted by the manufacturer ($z_{\text{sh}} > 4.0$ m), the accuracy drops and the corresponding fitting line slightly shifts above $y = x$.

To correct the sensor, we plot the following points: $(z_{\text{sh}}, z_{\text{sh}} - z_{\text{cor}})$. We then fit them with a polynomial $f\,(z_{sh})$ that minimises the error between z_{sh} and z_{cor} in the least squares sense (column (b) in Figs. 10 and 11). In practice, we found that an 8° polynomial adequately represents the set of points with a small error factor. The correct depth of z_{sh} is obtained by evaluating $f(z_{\text{sh}})$ for the whole depth map.

Owing to the limited number of discrete depth values that can exist in any point cloud output by Kinect,[35] we compute for each raw range value (z_{sh}) a respective corrected value (z_{cor}). As a result, the correction of the depth image is reduced to the correction of the known depth levels.[35]

The depth correction module runs in the GPU on a pixel by pixel basis. We apply the correction only on the valid depth readings before any further processing takes place to ensure a better quality

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press



(a) Before correction (RMSE = 0.0703m) (b) Error model (c) Result after correction (RMSE = 0.0538m)

Fig. 11. Good Kinect. (a) Before correction (RMSE = 0.0703 m). (b) Error model. (c) Result after correction (RMSE = 0.0538 m).

Table I. Symbols and their corresponding meaning.

Symbol	Quantity/unit/ range/size	Definition
N	5	Number of cameras covering the whole scene
$f_x,\ f_y$	Pixel	Focal lengths in pixels towards X, Y axes respectively
$c_x,\ c_y$	Pixel	Centre of imager
$\mathbf{R}$	3×3	Rotation transform from colour camera (RGB) frame to IR frame
$\mathbf{T}$	3×1	Translation describing the shift between RGB frame centre and IR frame
u, v	Pixel	Coordinates of a given pixel in IR frame
$z(u,\ v)$	0.8–4.5 (m)	Depth measurement (distance that separates the sensor from the scene)
x, y, z	Meter	Coordinates of robot in world frame
$\dot{x},\ \dot{y},\ \dot{z}$	Meter/second	Velocities of robot on the three axes of world frame
$\ddot{x}, \ddot{y}, \ddot{z}$	Meter/second^2	Accelerations of robot on the three axes of world frame
Im Size	640×480	Size of images in every RGB and depth frame
RGBD–K_i	640×480	Raw RGBD (RGB + Depth) data streamed by the ith ($0 \leq i \leq 4$) Kinect sensor
P–RGBD–K_i	640×480	Pre-processed and aligned frames
Pos_i	$3 + 3$	Position and orientation of the robot delivered by the ith ($0 \leq i \leq 4$) Kinect sensor
RF–Rob$_i$	$3 + 3$	Position and orientation of robot resulting from the robust H_∞ filter
Cov_i	3×3	Covariance matrix associated to RF–Rob$_i$
$\alpha,\ \beta,\ \gamma$	1×3	Orientations of robot

Table II. RMSE (m) for some Kinects before and after the correction.

	Kinect 0	Kinect 1	Kinect 3	Kinect 4
Before	0.1114	0.1474	0.2189	0.0703
After	0.0490	0.0598	0.0633	0.0538

of data. This allows us to fully benefit from the available accuracy of the sensor. After the correction (column (c) in Figs. 10 and 11), the depth data is almost the same as the ground truth. However, for greater range of values, the resolution of the sensor decreases and only some discrete measurements can be obtained. As we can see from the figures, the density of the samples that we used to compute the correction model decreases with increasing range. Table II shows the error in measurements for some of the Kinects used in our cooperative multiview tracking experiments.

8.3. The robust H_∞ filter

The results discussed in the following sections are obtained from the setup shown in Fig. 12.

Figures 13 and 14 present, respectively, the best and the worst performances of the robust H_∞ filter tracking algorithm along with comparative results when delivered by KF. As we have explained earlier, the motion model of the robot is unknown. Hence, we applied a generic Newtonian system to mimic its behaviour. Afterwards, we overcome the uncertainties with the robust H_∞ filter.

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

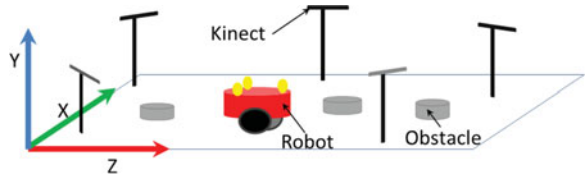


Fig. 12. *X*, *Y*, and *Z* axes in our experimental setup.

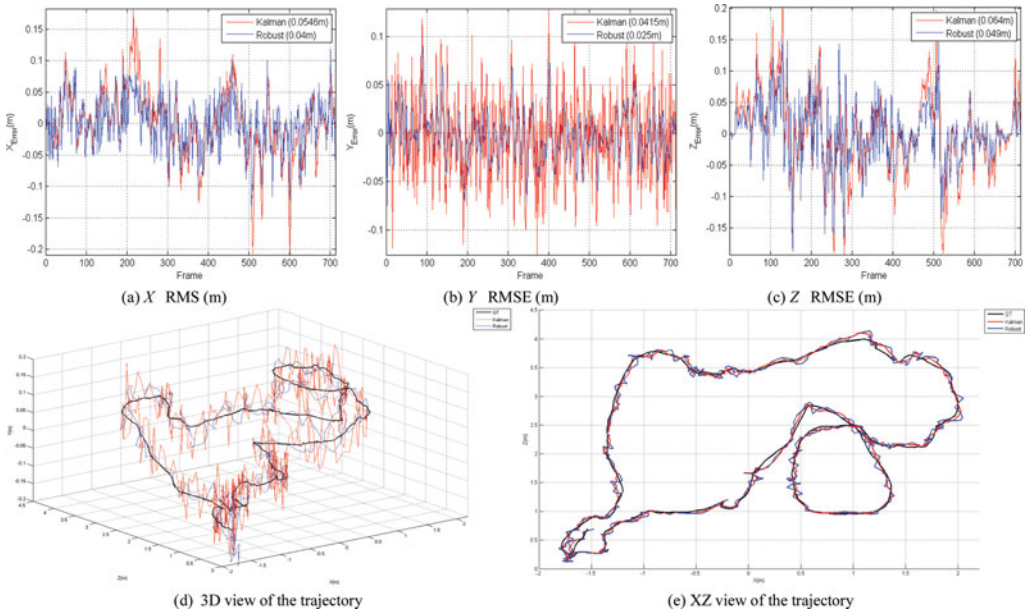


Fig. 13. The best tracking results after applying robust H_∞ and KF on *X*, *Y*, and *Z* coordinates. (a) *X* RMS (m). (b) *Y* RMSE (m). (c) *Z* RMSE (m). (d) 3D view of the trajectory. (e) *XZ* view of the trajectory.

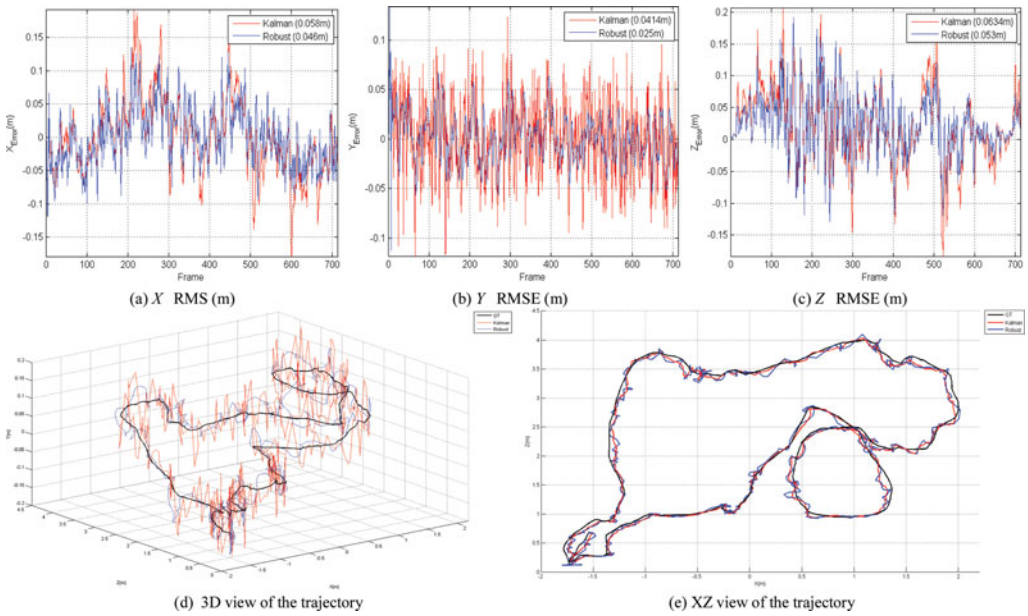


Fig. 14. The worst tracking results after applying robust H_∞ and Kalman filters on *X*, *Y*, and *Z* coordinates. (a) *X* RMS (m). (b) *Y* RMSE (m). (c) *Z* RMSE (m). (d) 3D view of the trajectory. (e) *XZ* view of the trajectory.

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

Table III. Error in *X*-component for all cameras.

X RMSE (m)	Kinect 0	Kinect 1	Kinect 2	Kinect 3	Kinect 4
KF	0.0570	0.0575	0.0580	0.0552	0.0546
RF	0.0433	0.0435	0.0456	0.0440	0.0403
Difference (KF – RF)	0.0137	0.0140	0.0124	0.0112	0.0143

Table IV. Error in *Y*-component for all cameras.

Y RMSE (m)	Kinect 0	Kinect 1	Kinect 2	Kinect 3	Kinect 4
KF	0.0392	0.0429	0.0414	0.0415	0.0415
RF	0.0253	0.0253	0.0253	0.0253	0.0252
Difference(KF – RF)	0.0139	0.0176	0.0161	0.0162	0.0163

Table V. Error in *Z*-component for all cameras.

Z RMSE (m)	Kinect 0	Kinect 1	Kinect 2	Kinect 3	Kinect 4
KF	0.0614	0.0594	0.0634	0.0590	0.0640
RF	0.0537	0.0524	0.0530	0.0534	0.0493
Difference (KF – RF)	0.0077	0.0070	0.0104	0.0056	0.0147

For X and Y coordinates (X, Y, and Z axes are shown in Fig. 12), Figs. 13 and 14 show almost similar error shape for the two filters: KF and robust H_∞. This happens because of the similarity in the model of motion for both the algorithms. Nevertheless, the tracking error with robust H_∞ is smaller. The smallest error within all five cameras for X-coordinate was 0.0403 m with RF against 0.0546 m for KF (Fig. 13(a)). The worst case in X was 0.0450 m for RF against 0.0580 m for KF (Fig. 14(a)). For Y-coordinate, the best RMSE was 0.0252 m with RF against 0.040 m for KF (Fig. 13(b)). The worst case in Y was 0.0253 m with RF against 0.0429 m for KF (Fig. 14(b)).

For the Z-component, the best result with RF was 0.0493 m against 0.0590 m for KF (Fig. 13(c)). The worst result was 0.053 m with RF against 0.0634 m for KF (Fig. 14(c)).

Throughout the experiments, the RF was less affected by the inaccuracies in the parameters of the system and always gave the best estimation. More importantly, it was able to predict the position of a moving robot even if no measurements were available. The detailed results for all the five sensors are given in Tables III–V.

The results shown in these tables are obtained after the correction of all cameras with their respective models. On the other hand, the effectiveness of some sensors against others is highly biased by the trajectory of the vehicle. If all the cameras have the same accuracy, the closest one to the robot will be the best candidate to precisely capture the position.

8.4. Covariance intersection

At the final stage of our cooperative multiview tracking pipeline, the CI filter is adapted to fuse the position data resulting from the sensor-wise estimates. To validate our finding about the CI weighting coefficients, we compared three different approaches of applying the weights on estimates. We tested the weighting with only the error in estimation (P_n) obtained by RF. Then we combined the latter with uncertainty in the accuracy of the sensors (P_n and K_n). Finally, we combined both the elements with the confidence in the depth measurement (P_n, K_n, and Z_n). After considering each new parameter affecting the process of data fusion, the quality of the estimation was improved. We tested our CI algorithm on both: the estimate of the trajectory resulting from the KF (CI + KF), and the one obtained from the robust H_∞ filter (CI + RF). The results of the fused trajectories were as follows.

Firstly, for the X-coordinate with P_n on its own (Fig. 15(a), Table VI (column X)) gave us an error of 0.028 m with RF, whereas with KF it gave an error of 0.0415 m. After considering the accuracy of the sensor (Fig. 16(a), Table VII (column X)), the error was reduced for both the filters to reach 0.0188 m with RF and 0.028 m for KF. The introduction of Z_n (Fig. 17(a), Table VIII (column X))

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

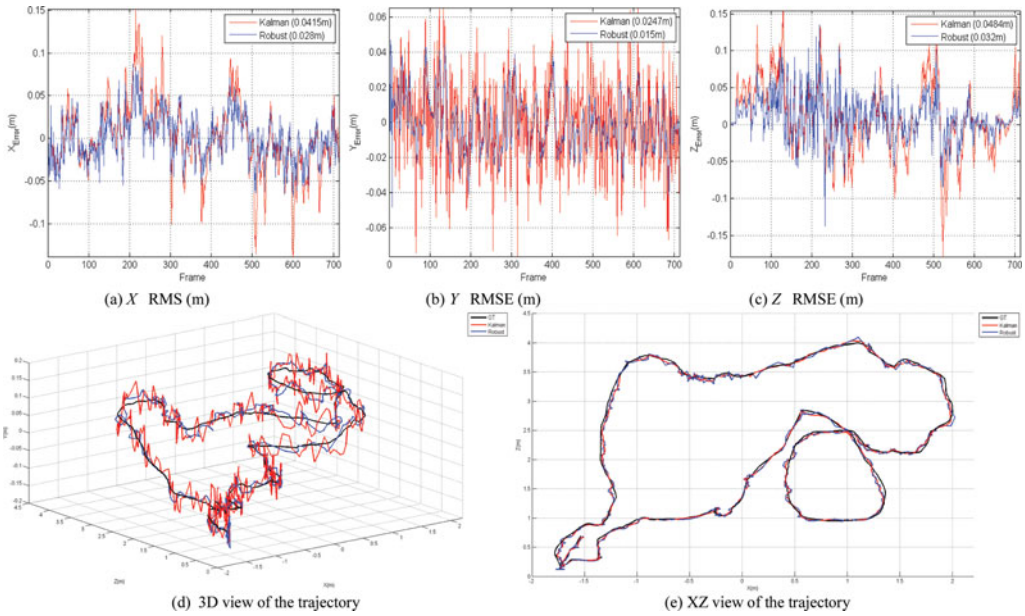


Fig. 15. P_n, weighting results. (a) X RMS (m). (b) Y RMSE (m). (c) Z RMSE (m). (d) 3D view of the trajectory. (e) XZ view of the trajectory.

Table VI. Final tracking error after CI filtering with P_n weighting.

Filters	X RMSE (m)	Y RMSE (m)	Z RMSE (m)
CI + KF	0.0415	0.0247	0.0484
CI + RF	0.0275	0.0154	0.0323
Difference CI + (KF – RF)	0.014	0.0093	0.0161

Table VII. Final tracking error after CI filtering with P_n, K_n weighting.

Filters	X RMSE (m)	Y RMSE (m)	Z RMSE (m)
CI + KF	0.0281	0.017	0.0333
CI + RF	0.0188	0.011	0.0223
Difference CI + (KF – RF)	0.0093	0.0065	0.011

Table VIII. Final tracking error after CI filtering with P_n,K_n, and $Z_n$weighting.

Filters	X RMSE (m)	Y RMSE (m)	Z RMSE (m)
CI + KF	0.0165	0.0097	0.0195
CI + RF	0.011	0.006	0.0129
Difference CI + (KF – RF)	0.0055	0.0037	0.0066

further approached the estimation to its ground truth counterpart as the error reached 0.011 m with RF and 0.016 m with KF.

Secondly, for the Y-coordinate, P_n on its own (Fig. 15(b), Table VI (column Y)) again gave us an error of 0.0154 m for RF, whereas with KF it gives 0.0247 m. After adding the accuracy parameter of the sensor, the error was reduced to 0.011 m with RF. Likewise, the introduction of K_n (Fig. 16(b), Table VII (column Y)) positively affected the accuracy of KF estimates as it increased to 0.017 m. Lastly, when we introduce Z_n (Fig. 17(b), Table VIII (column Y)), the error in estimation dropped to 0.006 m, at the same time the error with KF dropped to 0.0098 m.

Thirdly, for Z-component, P_n (Fig. 15(c), Table VI (column Z)) on its own again gave us an error of 0.0323 m for RF, whereas with KF it gave an error of 0.0484 m. After adding the accuracy parameter

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

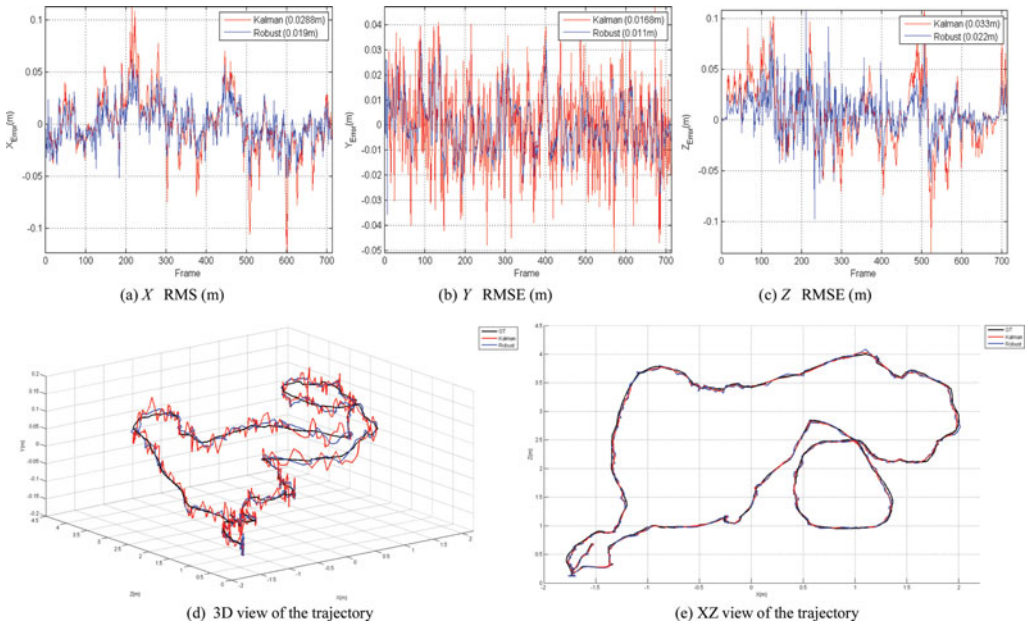


Fig. 16. P_n, K_n, and Z_n weighting results. (a) X RMS (m). (b) Y RMSE (m). (c) Z RMSE (m). (d) 3D view of the trajectory. (e) XZ view of the trajectory.

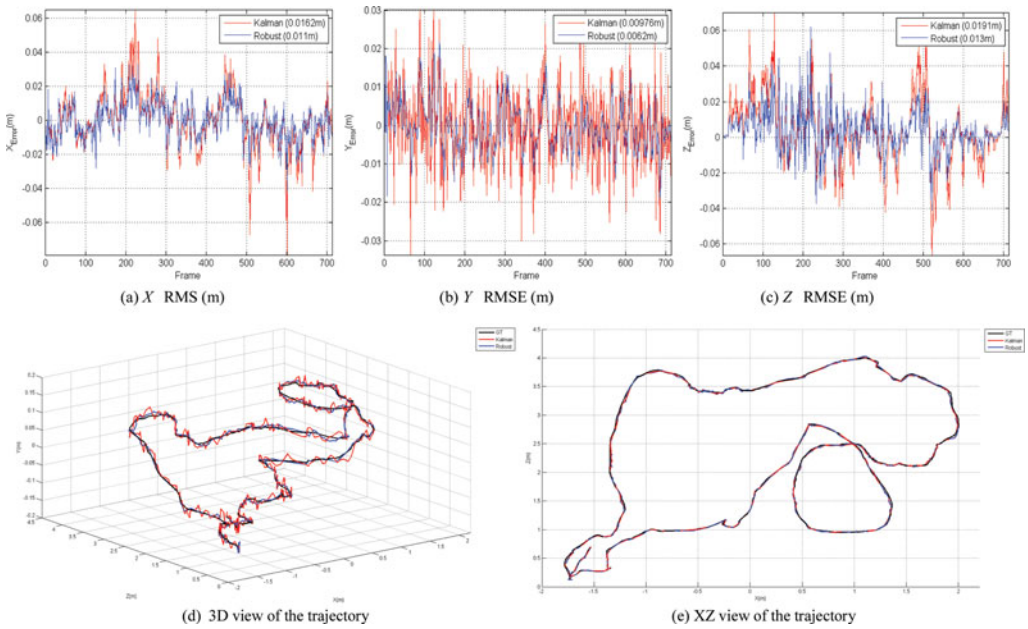


Fig. 17. P_n, K_n, and Z_n weighting results. (a) X RMS (m). (b) Y RMSE (m). (c) Z RMSE (m). (d) 3D view of the trajectory. (e) XZ view of the trajectory.

of the sensor, the error was slightly reduced to 0.0223 m with RF. In the same way, the introduction of K_n (Fig. 16(c), Table VII (column Z)) increased the accuracy of KF estimates to 0.0333 m. Z_n (Fig. 17(c), Table VIII (column Z)) positively affected the error in RF to reach 0.013 m; the error in KF estimation also decreased to reach 0.0195 m.

Based on the result obtained from our experiments, the quality of estimation between RF and KF is not significantly different. This is true because of the compensating effect of the CI algorithm through all the sensors. In other words, each sensor participates with its best estimation. After the correction, the most accurate measurement is used in both KF and RF to compute the next prediction.

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

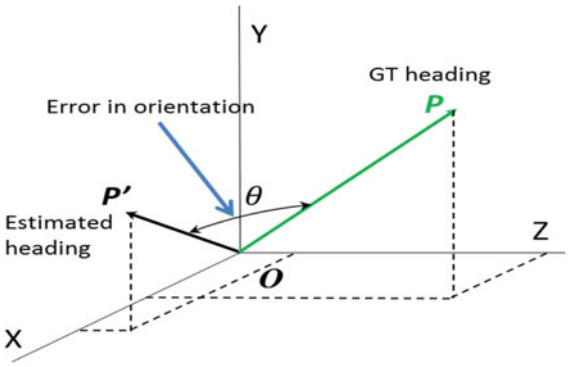


Fig. 18. Angle between the GT and the estimated heading.

As a consequence, the difference in the fused output is not significant when we attribute a higher weight to the most reliable measurement. In addition, given the limited space used for this indoor experiment, the RF performs theoretically just as the KF when the robot moves linearly according to a predefined motion model.

8.5. Orientation of vehicle

To test the orientation angles delivered by our solution, we compute error angle between the ground truth heading and the estimated one. To this end, we use the dot product between the two vectors representing the ground truth and the estimated direction of the robot. As we can see in Fig. 18, the dot product between the two vectors, $\vec{P}$ and $\overrightarrow{P'}$, can be obtained from their magnitude and the angle between them as shown in Eq. (50):

$$P \times P' = |P| \left|P'\right| \times \cos(\theta). \tag{50}$$

From Eq. (50), we get:

$$\cos(\theta) = \frac{P \times P'}{|P|\,|P'|}. \tag{51}$$

The error angle between the two directions becomes,

$$\theta = \arccos\left(\frac{P \times P'}{|P|\,|P'|}\right). \tag{52}$$

After applying Eq. (52), we got the results illustrated in Fig. 19. The error in the orientation of the vehicle resulting from RF was 11°, whereas KF-RMSE was 17°. With both the filters the error in the orientation of the mobile robot was not important. More importantly, proportional to the accuracy of the position of the vehicle, the application of CI better improved the accuracy of the orientation angle as follows: With P_n on its own, RF-RMSE was 7.1° and KF-RMSE became 12°. With P_n, K_n the results are even better where RF-RMSE became 4.8° and KF-RMSE was 8.1°. Finally, the introduction of Z_n significantly reduced the error to 2.7° for RF and 4.6° for KF. The result is very accurate, given the fact that we did not imply the three angles of orientation in the filtering algorithm. Consequently, our approach to compute the heading of the vehicle proved its effectiveness and high adequacy for real-time systems.

9. Conclusions and Future Works

In this work, we presented a novel approach to accurately track moving vehicles in indoor environments with cooperative multiple consumer RGBD cameras. We described the details of our methodology and findings. Our findings about the RGBD sensor correction are of importance for a more accurate measurement with any type of sensors suffering from the same drawbacks. To our

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

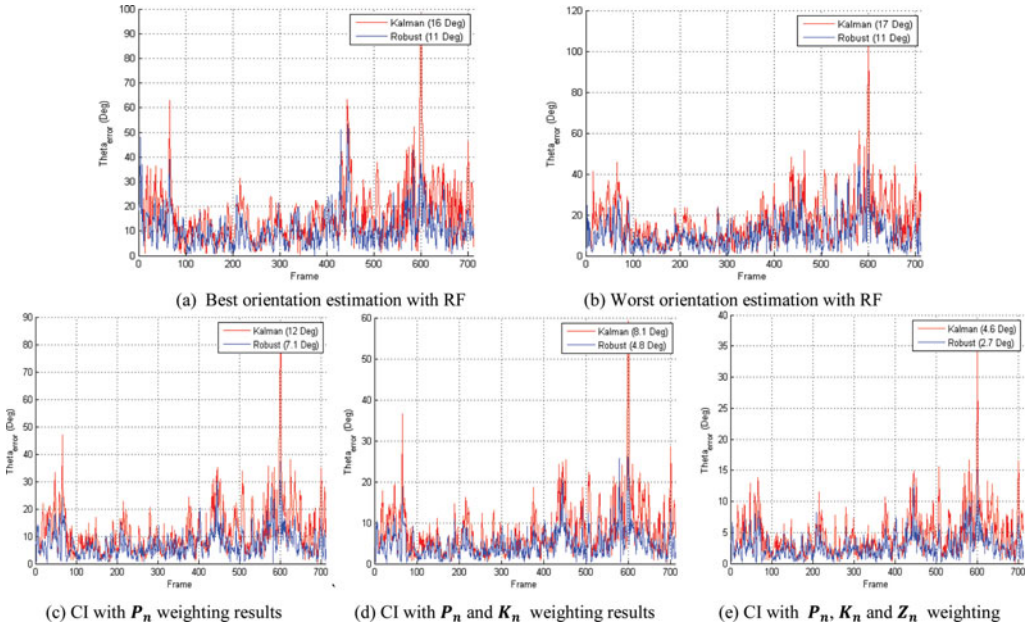


Fig. 19. Error in the estimated orientation of vehicle. (a) Best orientation estimation with RF. (b) Worst orientation estimation with RF. (c) CI with P_n weighting results. (d) CI with P_n and K_n weighting results. (e) CI with P_n, K_n, and Z_n weighting.

knowledge, we were the first to investigate and apply the robust filtering in objects tracking using RGBD sensor. We also demonstrated the power of the latter to overcome the lack of knowledge about the system governing the behaviour of vehicles. We considered the quality of measurements and the estimates provided by each sensor in the CI algorithm. We successfully combined all the single contributions of cameras in one consistent and cooperative output. Test results show the performance we achieved at a frame rate of 25 Fps with five Kinects. The innovative development based on the GPU for all the bottleneck stages of processing (capture and markers extraction) helped significantly to achieve the real-time performance. For robust filtering and CI algorithms, no parallelisation was required because these were just applied on some position data (five 3D points). Their linear computational nature required a powerful sequential processing, which was the asset for CPUs (their frequency of processing was much higher than that of GPUs).

In future work, we aim to overcome the multiple Kinect interference problems by algorithmic means using the stereo RGB/IR images to complete the missing depth information. We also aim to apply the same filtering scheme on 3D reconstruction applications for object recognition purposes. Another planned work is the combination of motion and 3D structure all together for a complete acquisition of the shape and behaviour of robots moving in the scene.

References

1. H. Yang, L. Shao, F. Zheng, L. Wang and Z. Song, "Recent advances and trends in visual tracking: A review," *Neurocomputing*, **74**(18), 3823–3831 (Nov. 2011).
2. K. Litomisky, "Consumer rgb-d cameras and their applications," *Tech. rep.* University of California, 2012.
3. J. Fung and S. Mann, "OpenVIDIA," *Proceedings of the 13th Annual ACM International Conference on Multimedia – MULTIMEDIA '05* (2005) pp. 849–852.
4. A. Amamra and N. Aouf, "Real-Time Robust Tracking with Commodity RGBD Camera," *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics* (2013) pp. 2408–2413.
5. A. Yilmaz, O. Javed and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.* **38**(4), 1–45 (2006).
6. M. Taj and A. Cavallaro, "Multi-view Multi-object Detection and Tracking," *IEEE Comput. Soc Studies in Computational Intelligence*, 263–280 (2010).
7. F. Lv, T. Zhao and R. Nevatia, "Self-calibration of a camera from video of a walking human," *Proceedings of the 16th International Conference on Pattern Recognition*, vol. 1 (2002), pp. 562–567.

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press

8. K. Okuma, A. Taleghani and N. De Freitas, "A Boosted Particle Filter: Multitarget Detection and Tracking," *Proceedings of the 8th European Conference on Computer Vision (ECCV),* Prague, Czech Republic (2004).
9. S. Z. Li, "Multi-Pedestrian Detection in Crowded Scenes: A Global View," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2012) pp. 3124–3129.
10. A. F. Bobick, S. S. Intille, J. W. Davis, F. Baird, C. S. Pinhanez, L. W. Campbell, Y. A. Ivanov, A. Schütte and A. Wilson, "The KidsRoom: A perceptually-based interactive and immersive story environment," *Presence Teleoperators Virtual Environ.* **8**(4), 369–393 (Aug. 1999).
11. S. Intille, J. Davis and A. Bobick, "Real-Time Closed-World Tracking," *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (1997) pp. 697–703.
12. C. Wren, A. Azarbayejani, T. Darrell and A. P. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE Trans. Pattern Anal. Mach. Intell.* **19**(7), 780–785 (Jul. 1997).
13. J. Flusser and T. Suk, "Rotation moment invariants for recognition of symmetric objects," *IEEE Trans. Image Process.* **15**(12) 3784–3790 (Dec. 2006).
14. S. Y. Chen, "Kalman filter for robot vision: A survey," *IEEE Trans. Ind. Electron.* **59**(11) 4409–4420 (Nov. 2012).
15. L. Liu, B. Sun, N. Wei, C. Hu and M. Q.-H. Meng, "A Novel Marker Tracking Method Based on Extended Kalman Filter for Multi-Camera Optical Tracking Systems," *Proceedings of the 5th International Conference on Bioinformatics and Biomedical Engineering* (2011) pp. 1–5.
16. Y. Rui and Y. Chen, "Better Proposal Distributions: Object Tracking Using Unscented Particle Filter," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (CVPR 2001), vol. 2 (2001), pp. II-786–II-793.
17. M. Pupilli and A. Calway, "Real-Time Camera Tracking Using a Particle Filter.," *British Machine Vision Conference (BMVC)*, 519–528 (2005).
18. D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. (Wiley-Interscience, New York, NY, 2006) 552 p.
19. W. Niehsen, "Information Fusion Based on Fast Covariance Intersection Filtering," *Proceedings of the Fifth International Conference on Information Fusion* (FUSION 2002), vol 2 (IEEE Cat. No. 02EX5997) (2002) pp. 901–904.
20. D. Smith and S. Singh, "Approaches to multisensor data fusion in target tracking: A survey," *IEEE Trans. Knowl. Data Eng.* **18**(12), 1696–1710 (Dec. 2006).
21. J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook and R. Moore, "Real-time human pose recognition in parts from single depth images," *Commun. ACM* **56**(1), 116 (Jan. 2013).
22. D. S. O. Correa, D. F. Sciotti, M. G. Prado, D. O. Sales, D. F. Wolf and F. S. Osorio, "Mobile Robots Navigation in Indoor Environments Using Kinect Sensor," *Proceedings of the 2012 Second Brazilian Conference on Critical Embedded Systems* (2012) pp. 36–41.
23. P. Henry, M. Krainin, E. Herbst, X. Ren and D. Fox, "RGB-D mapping: Using kinect-style depth cameras for dense 3D modeling of indoor environments," *Int. J. Rob. Res.* **31**(5), 647–663 (Feb. 2012).
24. T. Nakamura, "Real-Time 3-D Object Tracking Using Kinect Sensor," *Proceedings of the 2011 IEEE International Conference on Robotics and Biomimetics* (2011) pp. 784–788.
25. S. Izadi, A. Davison, A. Fitzgibbon, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges and D. Freeman, "Kinect Fusion," *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology – UIST '11* (2011) 559.
26. J. Tong, J. Zhou, L. Liu, Z. Pan and H. Yan, "Scanning 3D full human bodies using Kinects," *IEEE Trans. Vis. Comput. Graph.* **18**(4) 643–650 (2012).
27. J. Han, L. Shao, D. Xu and J. Shotton, "Enhanced computer vision with microsoft kinect sensor: A review," *IEEE Trans. Cybern.* **43**(5), 1318–1334 (Oct. 2013).
28. ROS Wiki, "kinect_calibration/technical - ROS Wiki" (online). Available at: http://wiki.ros.org/kinect_calibration/technical. Accessed: Jan. 27, 2014.
29. R. Reyes, I. Lopez, J. J. Fumero and F. de Sande, "accULL: A User-Directed Approach to Heterogeneous Programming," *Proceedings of the 2012 IEEE 10th International Symposium* on *Parallel and Distributed Processing with Applications*, (2012) pp. 654–661.
30. M. Andersen, T. Jensen, P. Lisouski, A. Mortensen, M. Hansen, T. Gregersen, and P. Ahrendt, "Kinect Depth Sensor Evaluation for Computer Vision Applications," Technical report, Dept. of Engineering, Aarhus University, Denmark, 2012.
31. M. Sedláček, "Evaluation of RGB and HSV Models in Human Faces Detection. *Central European Seminar on Computer Graphics*, Budmerice, Slovakia.
32. Y. S. Hung and F. Yang, "Robust H∞ Filtering with Error Variance Constraints for Discrete Time-Varying Systems with Uncertainty", *Automatica*, **39**(7), 1185–1194 (2003).
33. D. Franken and A. Hupper, "Improved Fast Covariance Intersection for Distributed Data Fusion," *Proceedings of the 7th International Conference on Information Fusion*, vol. 1 (2005), p 7.
34. K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors (Basel)* **12**(2), 1437–1454 (Jan. 2012).
35. A. Amamra and N. Aouf, "Robust and Sparse RGBD Data Registration of Scene Views," *Proceedings of the 17th International Conference on Information Visualisation* (2013) pp. 488–493.
36. R. Cucchiara, C. Grana, M. Piccardi, A. Prati and S. Sirotti, "Improving Shadow Suppression in Moving Object Detection with HSV Color Information," *Proc. IEEE Int',l Conf. Intelligent Transportation Systems*,

334–339 (Aug. 2001).
37. M. S. Mahmoud, "Resilient Linear Filtering of Uncertain Systems," *Automatica*, 40, 1797–1802 (2004).
38. L. Xie, L. Lu, D. Zhang, and H. Zhang, "Improved Robust H2 and H∞ Filtering for Uncertain Discrete-Time Systems" **40**(5), 873–880 (2004).
39. D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cogn. Process.* **12**(4), 319–340 (Nov. 2011).
40. J. Fung and S. Mann, "Using Graphic Devices in Reverse: GPU-Based Image Processing and Computer Vision," *Proceedings of the 2008 IEEE International Conference on Multimedia and Expo* (2008), pp. 9–12.
41. F. Jargstorff, "GPU Image Processing," *Proceedings of SIGGRAPH 2004* (2004).
42. E. Kilgariff and R. Fernando, "The GeForce 6 Series GPU Architecture," *ACM SIGGRAPH 2005 Courses* (2005).

https://doi.org/10.1017/S026357471400263X Published online by Cambridge University Press