Robotica, Volume 34, Issue 8, August 2016, pp. 1855-1879
DOI: 10.1017/S026357471400263X

Real-Time Multiview Data Fusion For Object Tracking With RGBD Sensors

Abdenour Amamra, Nabil Aouf
Centre For Electronic Warfare, Cranfield University, Defence Academy of the United Kingdom, Shrivenham, SN6 8LA
a.amamra@cranfield.ac.uk, n.aouf@cranfield.ac.uk

***Abstract*—This paper presents a new approach to accurately track a moving vehicle with a multiview setup of RGBD cameras. We first propose a correction method to eliminate a shift, which occurs in the 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 twenty five frames per second with a configuration of five cameras. Test results show the accuracy we achieved and the robustness of our solution to overcome the uncertainties in the measurements as well as the modelling.**

***Index Terms*—RGBD; real-time; tracking; multiview; Kalman filter; robust H_∞; covariance intersection; GPU.**

I. Introduction

NOWADAYS, real-time tracking of moving objects in 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 widely benefited from the 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 better perception of the surrounding space [2]. These cameras have the ability to deliver both the 3D map of the scene along with the corresponding colour image at a frequency of 30Fps.

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 a smoothing and a fusion algorithm. 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 the 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:

- We improve the raw measurements of the RGBD sensor using a mathematical correction model.
- We deal with the uncertainties in the model describing the motion of the vehicle using the Robust H_∞ filter, which has the ability to deal with the uncertainties in the measurements and the modelling.
- We apply the Covariance Intersection algorithm with adaptive weighting coefficients computed from the specific characteristics of our tracking setup.

The paper is structured as follows: in section II, we discuss the state of the art of image based tracking applications. We then explain the architecture of our system in section III. We give the details about the first two modules of the system in section IV. In section V, we detail the modelling of the uncertainties how the objects can be accurately tracked without prior knowledge of their motion. In section VI, we present the covariance intersection technique. In section VII, we clarify how it is possible to compute the orientation of the vehicle in our solution. We validate our finding in section VIII, where we plot the error graphs obtained from the real experiments. Finally, we discuss the possible improvements and future works in section IX.

II. 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]. Zhao et al. [7] presented a method for people tracking with a single camera. They used the 3D shape models of people that were projected back into the image space to perform the segmentation and to resolve the occlusions. Each human hypothesis is then tracked in 3D with a Kalman filter 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. Leibe et al. [9] present a pedestrian detection algorithm for crowded scenes. Their method iteratively aggregates local and global patterns for a better segmentation. These and other similar algorithms are challenged by the occluding and the partially occluding objects and appearance changes.

On the other hand, the cooperative multiview object tracking has recently benefited from a large interest against a single camera methods [6]. This advantage is largely driven by the wider coverage of the scene, which is an asset in handling occlusions. KidsRoom system developed at 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 the 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. Pfinder (Person-finder) 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 the 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 the 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 Kalman filter (KF) is an optimal solution for the class of linear Gaussian estimation problems [14]. For the nonlinear systems, a number of Bayesian techniques were proposed to perform the optimisation. When the Gaussian distribution is assumed, commonly used approaches include the extended Kalman filter (EKF) [15] and the Unscented Kalman filter (UKF) [16]. The particle filter is another numerical method that enables an approximate solution to the sequential estimation to be found. [17]. All the filters above 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 the real scenarios [18], the accuracy of the estimation is not the optimal. To overcome the uncertainty in the system, the Robust H_∞ filter (RF) is known for its ability to cope with the uncertainties impinging on the model and the measurements. The author is not aware of any RF system being used for accurate tracking. Furthermore, Covariance intersection (CI) [19] is applied to combine the estimates computed upon the raw output of each camera in a way that minimises the error in the global estimate [20].

RGBD sensors have recently become very popular in the capture of 3D data among professionals and researchers alike. They do not deliver just the colour but also the depth information at a frame rate of 30Fps. 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.

III. SYSTEM OVERVIEW

A. 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 30Hz and a resolution of 640 × 480 pixels. The sensor contains an 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 the 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 ($\boldsymbol{f_x}, \boldsymbol{f_y}, \boldsymbol{c_x}, \boldsymbol{c_y}$ intrinsic parameters for both RGB and IR cameras and the extrinsic parameters $[R, T]$ of the IR-RGB stereo setup) the actual accuracy of the capture may range from less than one millimetre 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 the measurements [24].

[1] http://www.xbox.com/en-GB/Kinect, 2012

TABLE 1
SYMBOLS AND THEIR CORRESPONDING MEANING

Symbol	Quantity/Unit/Range/ Size	Definition
$\boldsymbol{N}$	5	Number of cameras covering the whole scene
$\boldsymbol{f_x}, \boldsymbol{f_y}$	pixel	Focal lengths in pixel unit toward X, Y axes respectively
$\boldsymbol{c_x}, \boldsymbol{c_y}$	pixel	Centre of the imager
$\mathbf{R}$	3 × 3	Rotation transform from colour camera (RGB) frame to infrared (IR) frame
$\mathbf{T}$	3 × 1	The translation describing the shift between RGB frame centre and IR frame
$\boldsymbol{u}, \boldsymbol{v}$	pixel	Coordinates of a given pixel in the IR frame
$\boldsymbol{z(u,v)}$	0.8 – 4.5 (meter)	Depth measurement (the distant that separates the sensor from the scene)
$\boldsymbol{x, y, z}$	meter	Coordinates of the robot in the world frame
$\dot{\boldsymbol{x}}, \dot{\boldsymbol{y}}, \dot{\boldsymbol{z}}$	meter/second	Velocities of the robot on the three axes of the world frame
$\ddot{\boldsymbol{x}}, \ddot{\boldsymbol{y}}, \ddot{\boldsymbol{z}}$	meter/second^2	Accelerations of the robot on the three axes of the world frame
$\boldsymbol{Im\_Size}$	640 × 480	Size of the images in every RGB and Depth frame
$\boldsymbol{Rgbd\_K_i}$	640 × 480	Raw RGBD (RGB + Depth) data streamed by the *i-th* ($0 \leq i \leq 4$) Kinect sensor
$\boldsymbol{P\_Rgbd\_K_i}$	640 × 480	Pre-processed and aligned frames
$\boldsymbol{Pos_i}$	3 + 3	Position and orientation of the robot delivered by the *i-th* ($0 \leq i \leq 4$) Kinect sensor
$\boldsymbol{RF\_Rob_i}$	3 + 3	Position and orientation of the robot resulting from the *Robust* H_∞ filter
$\boldsymbol{Cov_i}$	3 × 3	Covariance matrix associated to RF_Rob_i
$\boldsymbol{\alpha, \beta, \gamma}$	1 × 3	Orientations of the robot

B. Hardware and software configuration

Our real-time tracking setup is composed of $N = 5$ Kinect cameras covering a volume of $4 \times 4 \times 3$ meters. All the sensors are connected to the same workstation (Figure 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 2GB 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, which adapts to every possible model of motion characterising the vehicle. From a software point of view, we need to access the output of all the Kinects simultaneously in real time. Thus, we used Kinect 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].

C. Real-time multi-Kinects tracking architecture

The system consists of four main modules through which flows every synchronised set of RGBD data streamed by the five sensors (see Figure 2, Figure 3):

1) Capture module

It streams the 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 the CPU and the 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 the 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 the 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:

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

2) Markers extraction module

To compute the position and the orientation of the robot, we fix three distinctive markers on its top (see Figure 1). The 3D pose of the 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

[2] http://www.mobilerobots.com/researchrobots/pioneerp3dx.aspx, 2013
[3] http://www.microsoft.com/en-gb/download/details.aspx?id=36998, 2013
[4] http://www.nvidia.com/object/cuda_home_new.html, 2013

module takes as input the aligned colour image and follows these steps:

- RGB to HSV (Hue, Saturation and Value) conversion. This conversion is motivated by the fact that the HSV space is more robust to the light intensity changes [31].
- Colour thresholding to separate the markers from the background.
- Erode and dilate the binary thresholded image.
- Actual extraction of the 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 the position and the orientation information. It acts on the whole trajectory of the 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 the Kinects with the reference frame to combine the data with the covariance intersection technique [33][19] following these steps:

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

As the markers can be occluded during tracking, it is necessary to fill the missing data. In addition, the quality of the 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 the 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.

IV. Capture and Markers Extraction

A. Capture module

1) Filtering the unreliable z_i

We filter the unreliable $\boldsymbol{z_i}$ 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.
- It exceeds the interval of useful depth data (0.8m – 4.5m).

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 the measurements among different Kinects covering the same scene. To this end, we compute a function that takes as input the shifted raw outputs of the 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 the disparity measurements. The origin of the problem is not related to the optical configuration of the infrared camera, but to an error factor generated by disparity computation module [34]. The latter is an offset that appears in the estimation of the distance between the sensor and the object. Its value becomes more important as the object becomes further away. Our purpose is to correctly remap the shifted depth ($\boldsymbol{z_{sh}}$) to its respective correct value. Hence, we take the depth measurements output by the sensor along with their correct ground truth counterparts ($\boldsymbol{z_{cor}}$), the pairs ($\boldsymbol{z_{sh}}$, $\boldsymbol{z_{cor}}$) are related by a function:

$$f(z_{sh}) = z_{sh} - z_{cor} \quad (1)$$

$\boldsymbol{f(z_{sh})}$ represents the shift from the correct reading. It computes for every depth value the corresponding error based on the ground truth reference ($\boldsymbol{z_{cor}}$) as shown in equation (1). $\boldsymbol{f(z_{sh})}$ is computed from the sample points taken simultaneously from Kinect and a high precision tracking system[5]. Once we obtain the pairs **($\boldsymbol{z_{sh}}$, $\boldsymbol{z_{cor}}$),** we fit the sparse data with a polynomial approximating the shape of the representative curve. The same polynomial will serve as a model, which takes as input a raw measurement ($\boldsymbol{z_{sh}}$) generated by the Kinect and outputs a corrected estimate ($\boldsymbol{z_{cor}}$). 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 (Figure 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 $\boldsymbol{x_i}$, $\boldsymbol{y_i}$ for every valid pixel coordinates (equations (2), (3)). To complete this step, we need the calibration parameters of the IR camera:

$$x_i = \left(u_{i\_ir} - c_{x\_ir} \right) z_i / f_{x\_ir} \quad (2)$$

$$y_i = \left(v_{i\_ir} - c_{y\_ir} \right) z_i / f_{y\_ir} \quad (3)$$

Afterwards, we apply the stereo calibration parameters [**R, T**] on the point $\boldsymbol{P(x_i, y_i, z_i)}$ to transform it from the IR

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

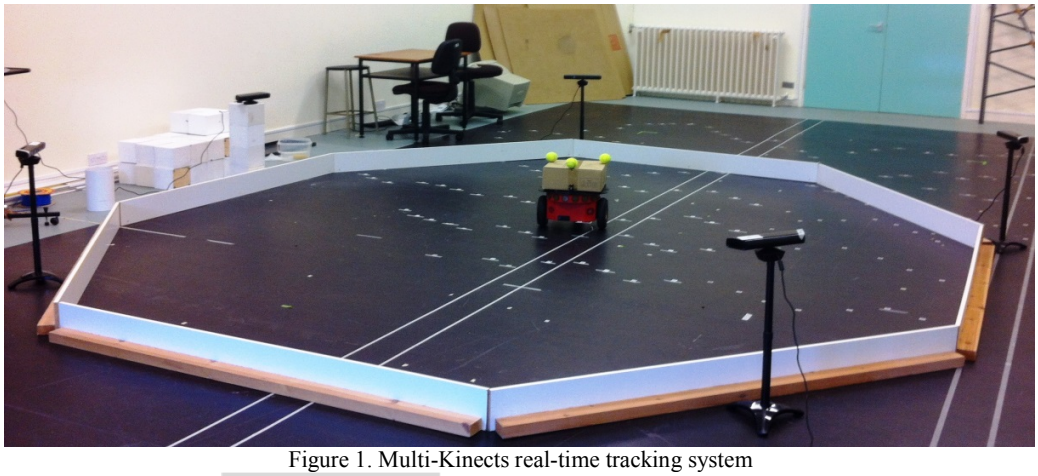


Figure 1. Multi-Kinects real-time tracking system

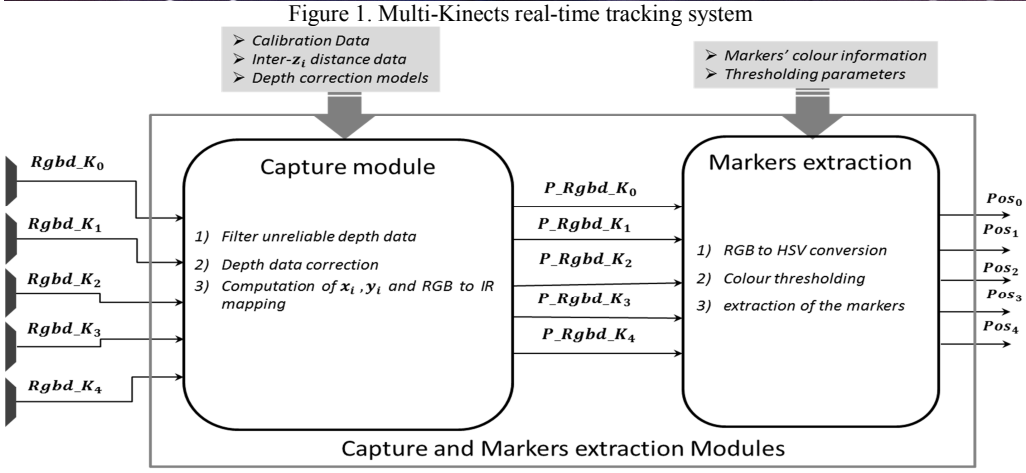


Figure 2. Capture and markers extraction modules

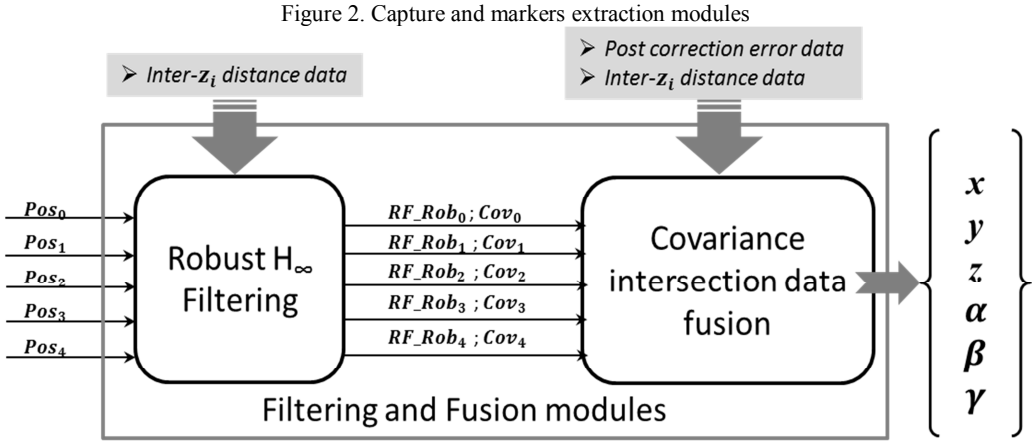


Figure 3. Filtering modules

coordinate system to the RGB frame $\boldsymbol{P'}(\boldsymbol{x'_i}, \boldsymbol{y'_i}, \boldsymbol{z'_i})$ (equation (4)):

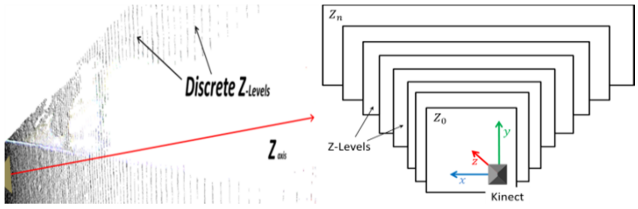
$$P' = \boldsymbol{R}P + \boldsymbol{T} \quad (4)$$

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

$$u_{i\_rgb} = \left(x'_i \ f_{x\_rgb} \ / \ z'_i \right) + \ c_{x\_rgb} \quad (5)$$

$$v_{i\_rgb} = \left(y'_i \ f_{y\_rgb} \ / \ z'_i \right) + \ c_{y\_rgb} \quad (6)$$

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



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

Figure 4 Kinect depth data

B. Markers extraction

1) RGB to HSV conversion

RGB colour model is the often used in computer and electronic systems. However, this coding has many downsides when we consider the 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 notice a large difference between the respective RGB combinations of the source and target objects. This even happens when a small change in the lighting occurs. On the other hand, HSV colour coding was proven to be less sensitive to the light intensity and the shadows [36].

2) Colour thresholding and the morphological operations

Once we convert our image to HSV space Figure 5 (a), we need to localise the areas of similar colour as the target. The thresholding and the binarisation Figure 5, are used to recognise and extract the three yellow markers from the scene. The alignment between the colour image and the depth map allows us to obtain the 3D coordinates for each marker. We apply erosion on the binary image to eliminate the 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 centres of mass. We start by extracting the contours of every marker in the binary image (white spots in Figure 5 (b)). Afterwards, we compute the zero-th and the first moments of each marker in the binary image [13] using equation (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 be a 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.

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

Figure 5. Markers extraction

V. ROBUST FILTERING

A. Motion model

Raw Kinect position measurements are only precise at close range from the sensor because the error in position remains below 5cms for 3 meters the correction module we introduced. It becomes below 5cms for up to 4.5m 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 the ordinary ground and flying robots is more stable and does not change in magnitude like it does in sign, because of the smooth and gradual variations of 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 $[\boldsymbol{x_k}\ \boldsymbol{y_k}\ \boldsymbol{z_k}\ \dot{\boldsymbol{x}}_{\boldsymbol{k}}\ \dot{\boldsymbol{y}}_{\boldsymbol{k}}\ \dot{\boldsymbol{z}}_{\boldsymbol{k}}]^{\boldsymbol{T}}$, and correct it accordingly after obtaining the measurements and the control input (the 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 $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ position of a given marker, the motion model will be:

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

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:

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

At every time-step $\boldsymbol{k}$; $\boldsymbol{s_k}$ is the estimated state of the vehicle (position and velocity); $\boldsymbol{t_k}$ is the measurement output by the sensor; $\boldsymbol{u_k}$: the accelerations of the vehicle along the three axes; $\boldsymbol{Q_k}$: the covariance of noise affecting the system ($\boldsymbol{w_k}$); $\boldsymbol{R_k}$: the covariance of noise affecting the measurements ($\boldsymbol{v_k}$).
From the equations (9), (10); the state-transition matrix $\boldsymbol{F}$ becomes (13).

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

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

B. Robust H_∞ filter

In the practical situations, the exact model of the system may not be available. The performance of such system becomes an important issue. The robust H_∞ filter [18], adopts as process and measurements models the state space representation in (15). In our case, the matrices of such models are defined by (13), (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 $\boldsymbol{k}$; $\boldsymbol{w_k}$ and $\boldsymbol{v_k}$ are uncorrelated zero-mean white noise processes with the covariance matrices $\boldsymbol{Q_k}$ and $\boldsymbol{R_k}$ respectively. The matrices $\Delta\boldsymbol{F_k}$ and $\Delta\boldsymbol{H_k}$ represent the uncertainties in the system and the measurements matrices. These uncertainties are assumed to be of the form:

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

$\boldsymbol{M_{1k}}, \boldsymbol{M_{2k}}$ and $\boldsymbol{N_k}$ are three known matrices, and $\boldsymbol{\Gamma_k}$ is an unknown matrix satisfying the bound:

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

It is assumed that $\boldsymbol{F_k}$ is non-singular. This assumption is not too restrictive; $\boldsymbol{F_k}$ should 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 $\tilde{\boldsymbol{F}}_{\boldsymbol{k}}$ should be less than one in magnitude).
- The estimator error satisfies the following worst-case bound:

$$\max_{w_k, v_k} \frac{\|\tilde{s}_k\|_2}{\|w_k\|_2 + \|v_k\|_2 + \|\tilde{s}_0\|_{O_1^{-1}} + \|s_0\|_{O_2^{-1}}} \leq \frac{1}{\theta} \tag{19}$$

- The estimation error of $\tilde{\boldsymbol{s}}_{\boldsymbol{k}}$ satisfies the following RMS bound:

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

The solution of this problem can be achieved by the following procedure:
1) Choose a scalar sequence $\boldsymbol{\alpha_k > 0}$ and a small $\boldsymbol{\varepsilon > 0}$.
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 $\boldsymbol{P_k}$ and $\tilde{\boldsymbol{P}}_{\boldsymbol{k}}$ as follows:

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

$\boldsymbol{O_1}, \boldsymbol{O_2}$ are the initial values we attribute to the estimation error covariance matrices for the computation of $\boldsymbol{R_{1k}}, \boldsymbol{R_{2k}}, \boldsymbol{F_{1k}}, \boldsymbol{H_{1k}}$ and $\boldsymbol{T_k}$. 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 $\boldsymbol{P_k}$ and $\tilde{\boldsymbol{P}}_{\boldsymbol{k}}$ satisfying the following Riccati equations:

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

$$\begin{aligned} \tilde{P}_{k+1} = {} & F_k \tilde{P}_k F_k^T + F_k \tilde{P}_k N_k^T \left(\alpha_k I \right. \\ & \left. - N_k \tilde{P}_k N_k^T\right)^{-1} N_k \tilde{P}_k F_k^T + R_{11k} + \varepsilon I \end{aligned} \tag{24}$$

Where the matrices $\boldsymbol{R_{1k}}, \boldsymbol{R_{2k}}, \boldsymbol{F_{1k}}, \boldsymbol{H_{1k}}$ and $\boldsymbol{G_k}$ are defined as:

$$R_{1k} = (\tilde{P}_k^{-1} - N_k^T N_k / \alpha_k)^{-1} F_k^T \tag{25}$$
$$R_{2k} = R_{1k}^{-1} (\tilde{P}_k^{-1} - N_k^T N_k / \alpha_k)^{-1} R_{1k}^{-T} \tag{26}$$
$$F_{1k} = F_k + R_{11k} R_{1k}^{-1} \tag{27}$$
$$H_{1k} = H_k + R_{12k}^T R_{1k}^{-1} \tag{28}$$
$$G_k = (P_k^{-1} - \theta^2 I)^{-1} \tag{29}$$

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 equation (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 $\boldsymbol{\varepsilon}$ is generally chosen as a very small positive number. In our case it was fixed to $\boldsymbol{\varepsilon = 10^{-8}}$.
The parameter $\boldsymbol{\alpha_k}$ has to be chosen large enough so that the conditions of equations (30), (31) are satisfied. However, when $\boldsymbol{\alpha_k}$ increases $\boldsymbol{P_k}$ also increases, which results in a looser bound for the RMS estimation error [18].
A steady-state of the robust filter can be obtained by letting the parameter $\boldsymbol{P_{k+1} = P_k}$ and $\boldsymbol{\widetilde{P}_{k+1} = \widetilde{P}_k}$ in equations (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 the uncertain system parameters. This asset enables us to track the 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 to the ground truth measurements show the effictivness of our approach against the naïve filtering scheme (Kalman filter does not consider the uncertainties in the system). The latter does not have the ability to cope with the 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.

VI. 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 the vehicles. At this level of our pipeline, we propose to develop the cooperation between multiple Kinects with the application of covariance intersection 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.

A. Covariance intersection Filtering

Based on the estimates determined by the Robust H_∞ filter $\boldsymbol{\hat{x}_{kn}}$ $\boldsymbol{(0 \le n \le N)}$ at time step $\boldsymbol{k}$, and their respective covariance matrices $\boldsymbol{P_{kn}}$; we compute a combined estimate $\boldsymbol{\tilde{x}}$ with its error covariance $\boldsymbol{P}$ where the true state of the system is $\boldsymbol{x}$ (real position of the vehicle).
If we consider $\boldsymbol{N}$ unbiased estimates related to each camera $\boldsymbol{\hat{x}_1, \hat{x}_2, \hat{x}_3 \ldots . \hat{x}_N}$ for the unknown state vector $\boldsymbol{\tilde{x}}$:

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

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

In the presence of a correlation between the estimation errors, the estimated $\boldsymbol{P}$ may become far too optimistic and this can cause divergence in sequential filtering. A conservative estimate can be given by applying the covariance intersection 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 nonnegative coefficients ω_n verifying the following consistency condition:

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

An estimate can always be obtained with:

$$P \ge P_0 \coloneqq E[(\tilde{x} - x)(\tilde{x} - x)^T] \tag{40}$$

Where $\boldsymbol{P \ge P_0}$ denotes the fact that $\boldsymbol{P - P_0}$ is positive semi-definite. Consequently, the coefficients ω_n are 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, Neihsen [19] has proposed a fast approximate solution.
For $\boldsymbol{trace(P_n) \le trace(P_m); 1 \le n, m \le N}$ one would expect $\boldsymbol{\omega_n \ge \omega_m}$.
From a computationally optimal point of view, rather than using the estimation uncertainty $\boldsymbol{P_n}$ the authors in [33] introduced the estimation certainty by considering $\boldsymbol{S_n = P_n^{-1}}$:

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

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

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

B. *Covariance intersection for multikinect tracker*

As we 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 the appropriate correction model. The mathematical formulation is as follows:

For the processing thread of the *n*-th camera:

$\boldsymbol{P_n}$: is the covariance matrix of the error in the estimate delivered by the Robust H_∞ filter.

$\boldsymbol{K_n}$: is the covariance matrix characterising the residual error after correction.

$\boldsymbol{Z_n}$: 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 is, the more accurate the resulting measurement becomes.

Figure 6 depicts the situation where every sensor has its native hardware accuracy matrix $\boldsymbol{K_n}$ (Red circles). Moreover, depending on the distance separating the camera from the target ($\boldsymbol{Z_n}$), we introduce the weighting coefficient $\boldsymbol{Z_n}$:

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

$$tr(K_n) + tr(P_n) + Z_n \leq tr(K_m) + tr(P_m) + Z_m \Rightarrow \omega_n \geq \omega_m \; ; \quad 1 \leq n, m \leq N \tag{43}$$

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

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

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

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

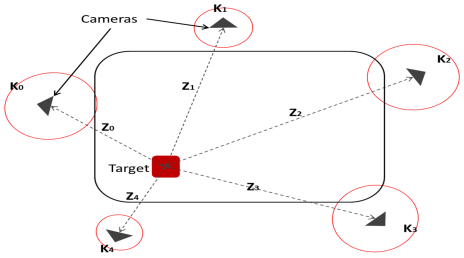


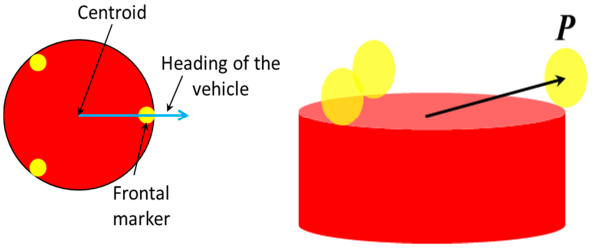
Figure 6. Covariance intersection parameters

Consequently, we only need to compute the traces of the matrices once. The denominator $\sum_{i=1}^{N}(tr(K_i) + tr(P_i) + Z_i)$ is also computed once. Afterwards, we subtract the corresponding parameter $(tr(K_n) + tr(P_n) + Z_n)$ 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 the equation (45) is more realistic and suitable for the real tracking scenarios.

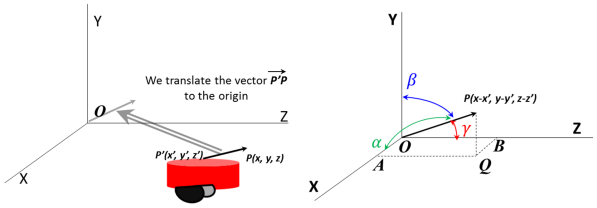
VII. ORIENTATION COMPUTATION

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 the 6DoF without any need to process the 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 the 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 the centroid, we can easily compute the remaining three orientation components ($\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$) by applying some simple trigonometry on the accurate positions of the centroid and the frontal marker (Figure 7 (a)(b)). This method is effective as it avoids us complicating the filter with the extra load of computations regarding the orientation.



(a) Heading vector (upper view) (b) Heading vector (side view)



(c) Pose of the robot in the scene (d) Orientation definition

Figure 7. Orientation computation.

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

$$\alpha = arcos\left(\frac{x-x'}{Mag}\right) \quad (47)$$

$$\beta = arcos\left(\frac{y-y'}{Mag}\right) \quad (48)$$

$$\gamma = arcos\left(\frac{z-z'}{Mag}\right) \quad (49)$$

Figure 7 (c), (d) illustrates how the problem of defining the orientations can be formulated. From the equations (47), (48), (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.

VIII. Results and Discussion

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

A. 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 blocs 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 bloc are processed simultaneously in the same GPU thread bloc. 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 the heterogeneous algorithms aims at a higher occupancy of the processors and full usage of the bandwidth when exchanging data between the central memory (RAM) and the global memory of the GPU (GMGPU)[41]. To this end, we focus on 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.

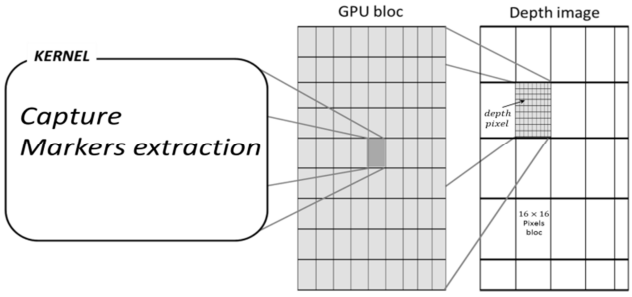
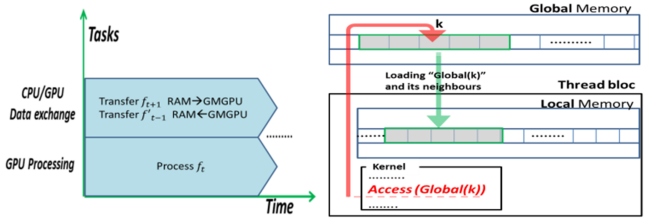


Figure 8.Kalman filter GPU implementation for depth map filtering



(a) RAM/GMGPU transfer (b) GMGPU→ RAM Data loading

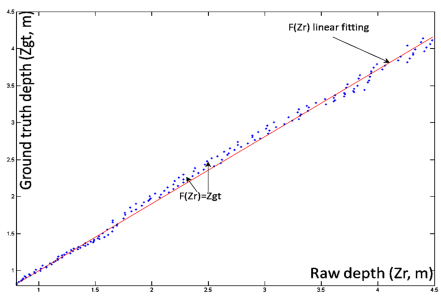
Figure 9. Optimisation of data exchange in the GPU

We therefore benefit from this idleness to exchange data. In other words, the following frame ($\boldsymbol{f_{t+1}}$) is sent from the RAM to the GMGPU, and the already available result ($\boldsymbol{f'_{t-1}}$) is sent back to the RAM. Simultaneously, the current frame ($\boldsymbol{f_t}$) is being processed on the device (GPU) (Figure 9(a)).

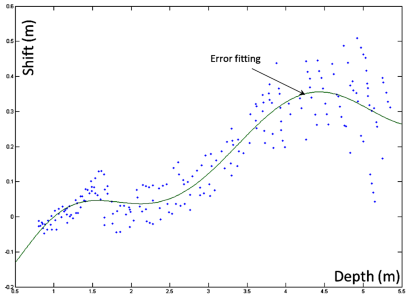
Memory coalescing: The GPU automatically loads the content of adjacent memory cells because its internal design assumes that it is highly probable for neighbouring data within the same area to be sooner requested as well [42]. Memory coalescing is another optimisation that significantly helps to increase the probability of threads in same warp (a group of 32 threads from the same thread 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.

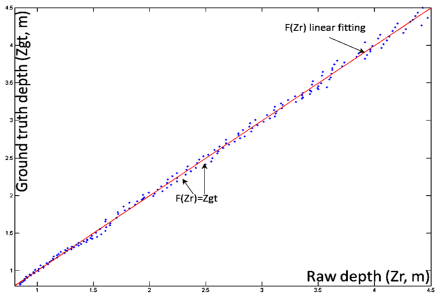
The CPU and the GPU are significantly different. A GPU can handle large amounts of data in many streams, performing relatively simple operations, but it is inadequate for heavy processing on a single or few streams. A CPU is much faster on a per-core basis and can perform complex operations on a single or few streams of data more easily. Consequently, the robust filter and the covariance intersection 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 the 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



(a) Before correction (RMSE = 0.2189m)

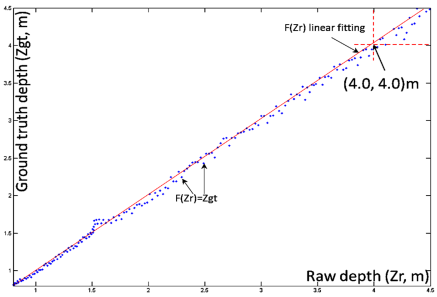


(b) Error model

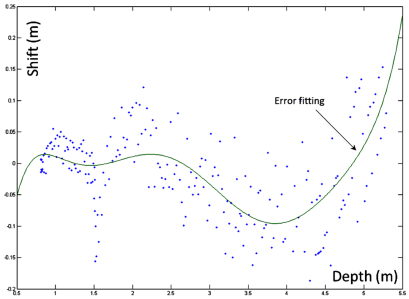


(c) Result after correction (RMSE = 0.0633m)

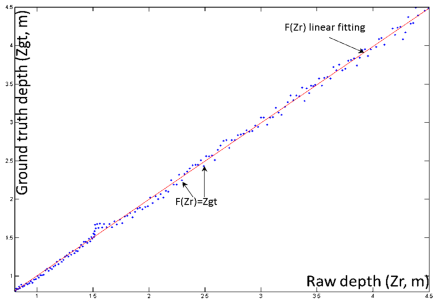
Figure 10. Drifty Kinect



(a) Before correction (RMSE = 0.0703m)



(b) Error model



(c) Result after correction (RMSE = 0.0538m)

Figure 11. Good Kinect

then executed on the resulting estimates to find the correct pose of the vehicle.

B. 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. Figure 10 and Figure 11 illustrate two cases where a drifty and a healthy sensor were adjusted using an eight degree polynomial. The first column of both figures ((a) before correction) presents the graphs of the function $\boldsymbol{g(z_{sh}) = z_{cor}}$. ($\boldsymbol{z_{sh}}$: shifted depth; $\boldsymbol{z_{cor}}$: ground truth depth). The more the fitting line approaches $\boldsymbol{y = x}$, the more accurate the sensor will be (the range measurement delivered by the Kinect is closer to the ground truth). With the faulty sensor (Figure 10), the curve representing $\boldsymbol{g(z_{sh})}$ is clearly shifted below $\boldsymbol{y = x}$. This happens because the sensor overestimates the range of the 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 (0.8m to 4.0m). On the other hand, with the good sensor (Figure 11), the representative curve is almost superimposed on $\boldsymbol{y = x}$ when the depth of the object is below 4.0m. When the object goes further than permitted by the manufacturer ($\boldsymbol{z_{sh}}$ > 4.0m), the accuracy drops and the corresponding fitting line slightly shifts above $\boldsymbol{y = x}$.

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

Because of the limited number of discrete depth values that can exist in any point cloud output by the Kinect [35], we compute for each raw range value $\boldsymbol{(z_{sh})}$ a respective corrected value $\boldsymbol{(z_{cor})}$. 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 of data. This allows us to fully benefit from the available accuracy of the sensor. After the correction (column (c) in Figure 10, Figure 11), the depth data is almost the same as the ground truth. However, for greater range 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 we used to compute the correction model decreases with increasing range. **TABLE 2** shows the error in measurements for some of the Kinects used in our cooperative multiview tracking experiments:

	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

TABLE 2. RMSE (m) for some Kinects before and after the correction

C. The Robust H_∞ filter

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

Figure 13, Figure 14 present respectively the best and the worst performances of the Robust H_∞ (**RF**) tracking algorithm along with comparative results when delivered by the

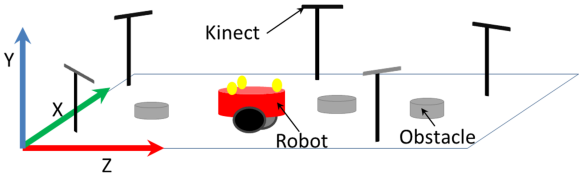
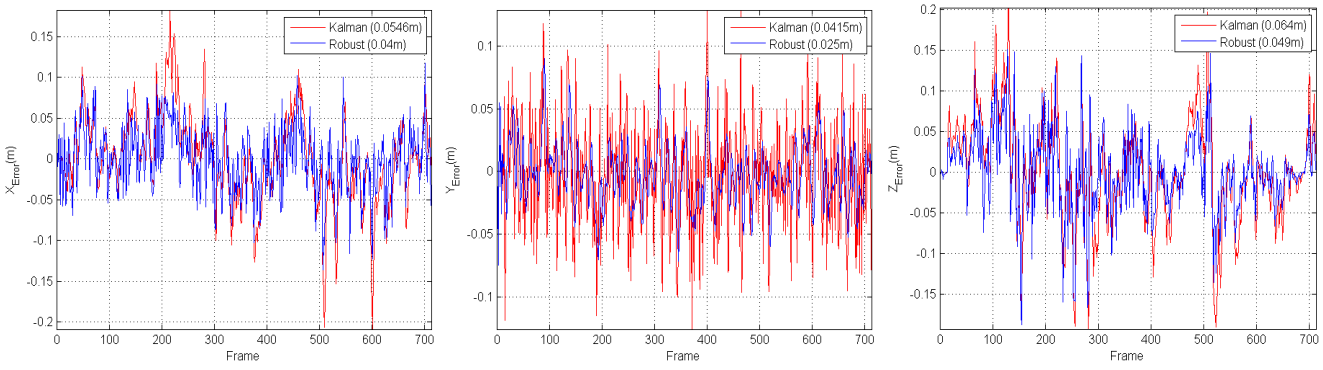
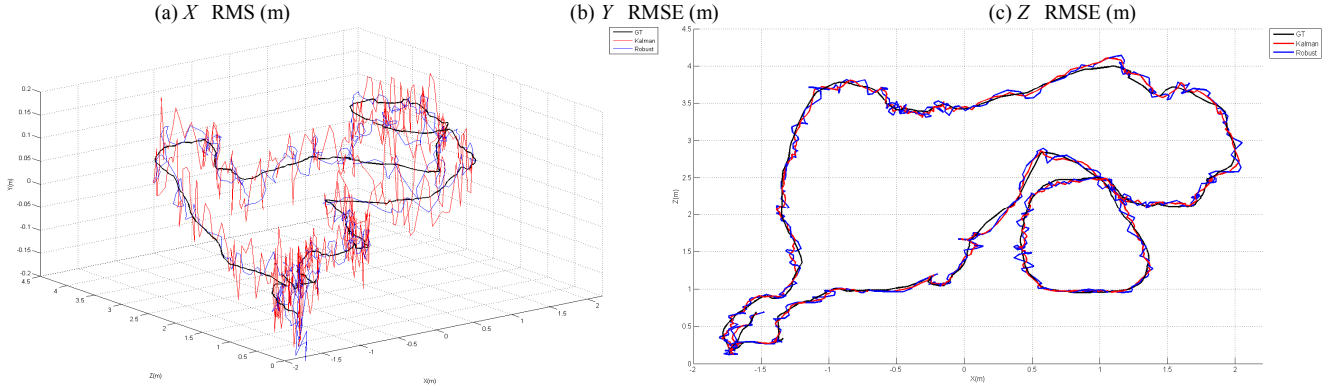


Figure 12. X,Y and Z axes in our experimental setup

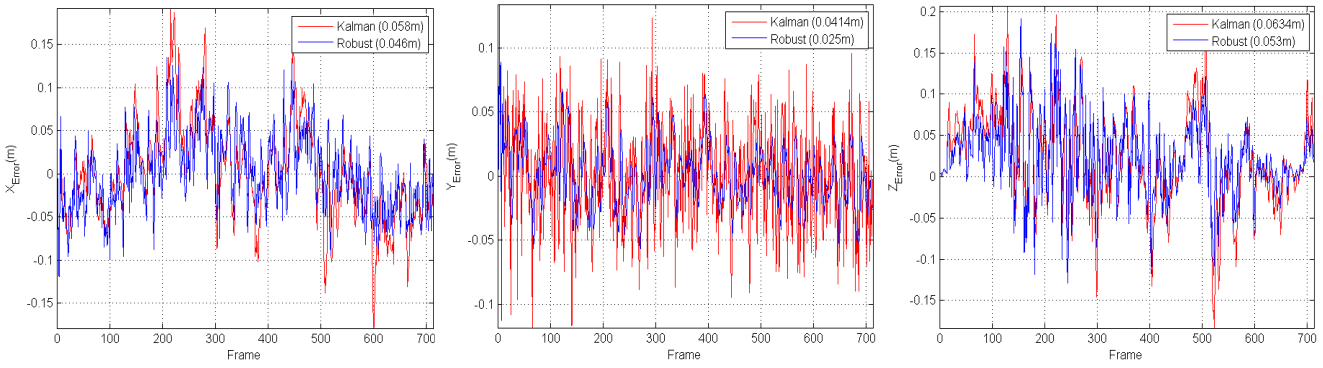


(a) X RMS (m) (b) Y RMSE (m) (c) Z RMSE (m)

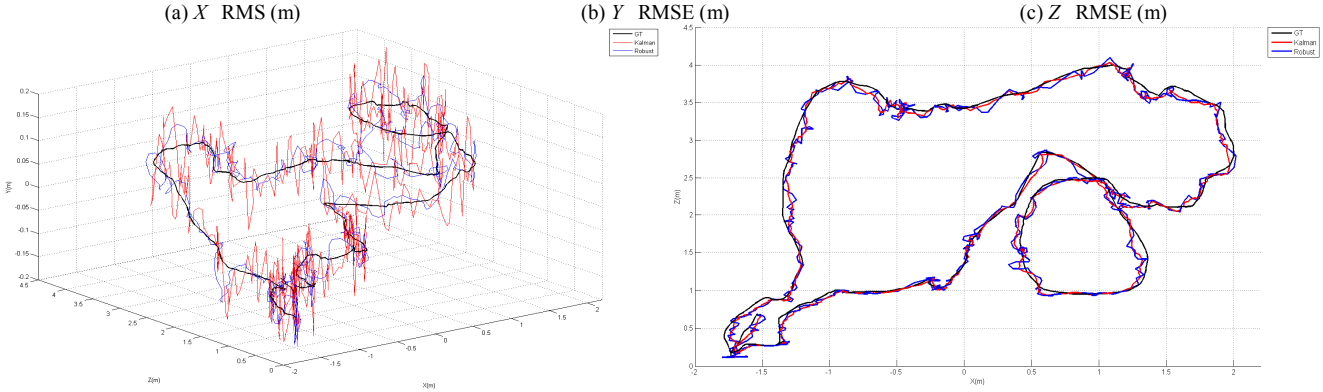


(d) 3D view of the trajectory (e) XZ view of the trajectory

Figure 13. The best tracking results after applying Robust $\boldsymbol{H}_\infty$ 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

Figure 14. The worst tracking results after applying Robust $\boldsymbol{H}_\infty$ and Kalman filters on X,Y and Z coordinates

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 3. Error in *X* 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 4. Error in *Y* 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

Table 5. Error in *Z* component for all cameras

Kalman filter (**KF**). As we 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.

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

For $\boldsymbol{Z}$ component the best result with RF was **0.0493m** against **0.0590m** for Kalman filter (Figure 13(c)). The worst result was **0.053m** with RF against **0.0634m** for KF (Figure 14(c)).

Throughout all the experiments, the RF was the less affected by the inaccuracies in the parameters of the system and always gives the best estimation. More importantly, it was able to predict the position of the moving robot even if no measurements were available. The detailed results for all the five sensors are given in TABLE 3, TABLE 4, TABLE 5.

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.

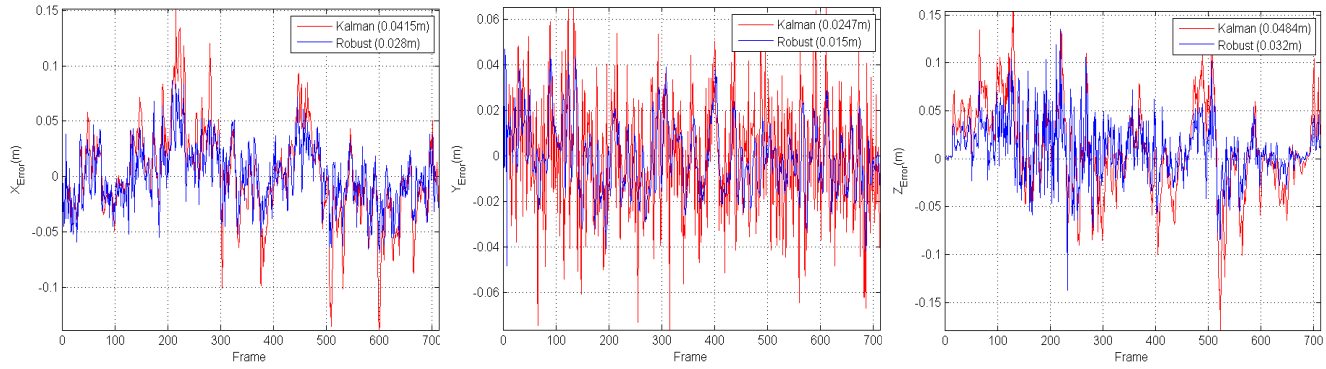
D. Covariance intersection (CI)

At the final stage of our cooperative multiview tracking pipeline, the covariance intersection filter (**CI**) 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 the estimates. We tested the weighting with only the error in estimation ($\boldsymbol{P_n}$) obtained by RF. Then we combined the latter with the uncertainty in the accuracy of the sensor ($\boldsymbol{P_n}$ and $\boldsymbol{K_n}$). Finally, we combined both elements with the confidence in the depth measurement ($\boldsymbol{P_n}$, $\boldsymbol{K_n}$ and $\boldsymbol{Z_n}$). 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 Kalman filter (CI + KF) and the one obtained from the robust H_∞ filter (CI + RF). The results of the fused trajectories were as follows:

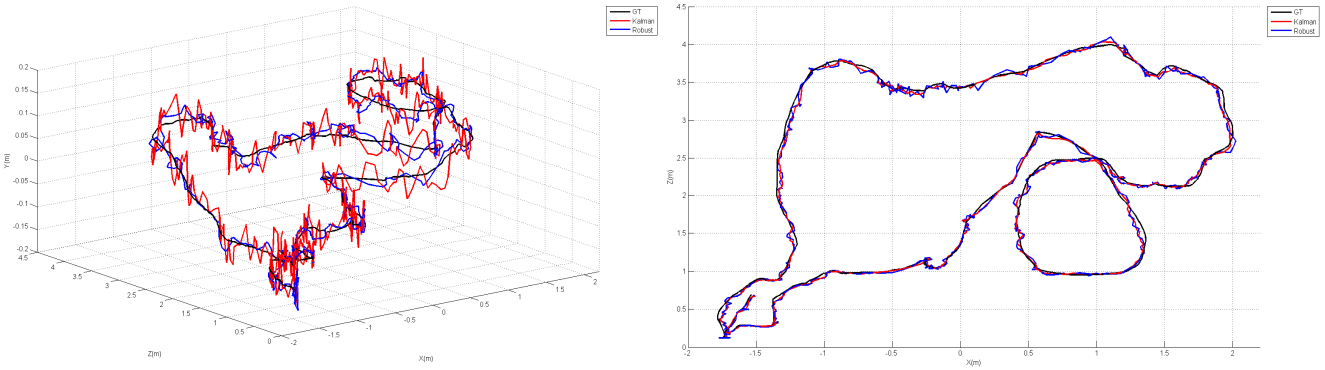
Firstly, for the $\boldsymbol{X}$ coordinate with $\boldsymbol{P_n}$ on its own (Figure 15.(a),TABLE 6 (col $\boldsymbol{X}$)) gave us an error of **0.028m** with RF, whereas with KF it gives **0.0415m**. After considering the accuracy of the sensor (Figure 16.(a), TABLE 7(col $\boldsymbol{X}$)), the error was reduced for both filters to reach **0.0188m** with RF, and **0.028m** for KF. The introduction of $\boldsymbol{Z_n}$ (Figure 17.(a), TABLE 8(col X)) further approached the estimation to its ground truth counterpart as the error reached **0.011**m with RF and **0.016m** for KF.

Secondly, for the $\boldsymbol{Y}$ coordinate $\boldsymbol{P_n}$ on its own (Figure 15.(b), TABLE 6(col Y)) gave us again an error of **0.0154m** for RF, whereas with KF it gives **0.0247m**. After adding the accuracy parameter of the sensor, the error was reduced to **0.011m** with the RF. Likewise, the introduction of $\boldsymbol{K_n}$(Figure 16.(b) , TABLE 7(col Y)) positively affected the accuracy of KF estimates as it increased to **0.017m**. Lastly, when we introduce $\boldsymbol{Z_n}$ (Figure 17.(b), TABLE 8 (col Y)) the error in estimation dropped to **0.006m**, at the same time the error with KF dropped to **0.0098m**.

Thirdly, for the $\boldsymbol{Z}$ component, $\boldsymbol{P_n}$ (Figure 15.(c), TABLE 6(col Z)) on its own gave us again an error of **0.0323m** for RF, whereas with KF it gives **0.0484m**. After adding the accuracy parameter of the sensor, the error was slightly reduced to **0.0223m** with RF. In the same way, the introduction of $\boldsymbol{K_n}$ (Figure 16.(c), TABLE 7(col Z)) increased the accuracy of KF estimates to **0.0333m**. $\boldsymbol{Z_n}$ (Figure 17.(c), TABLE 8(col Z)) positively affected the error in RF to reach **0.013m**, the error in KF estimation also decreased to reach **0.0195m**.

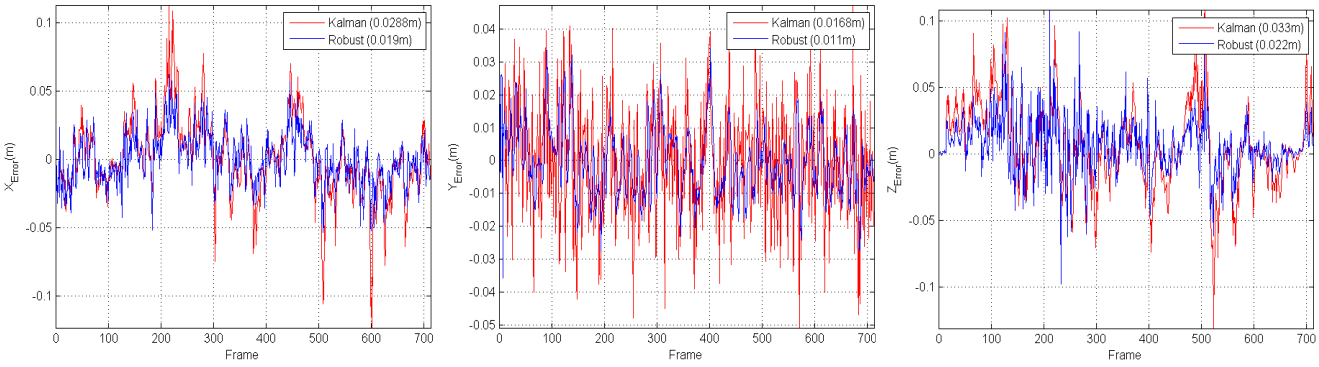


(a) X RMS (m) (b) Y RMSE (m) (c) Z RMSE (m)

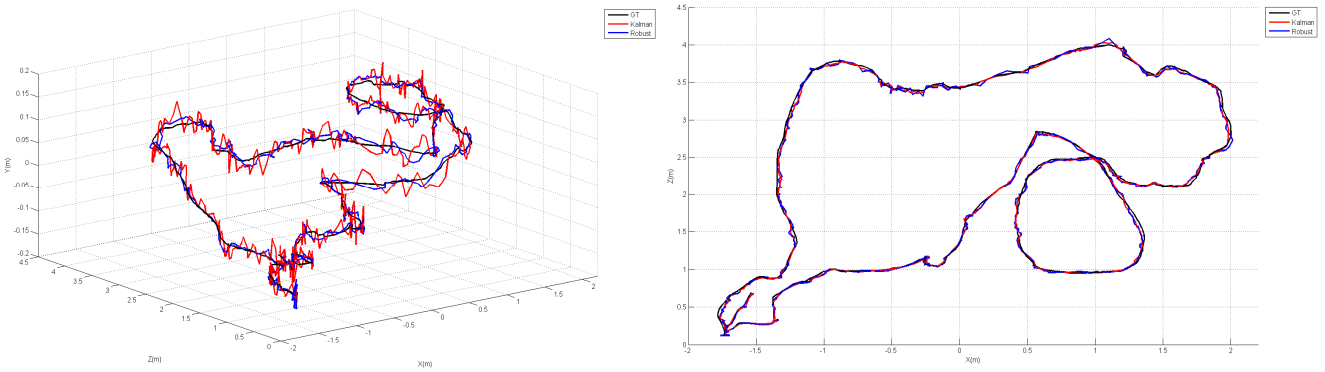


(d) 3D view of the trajectory (e) XZ view of the trajectory

Figure 15. $\boldsymbol{P_n}$ Weighting results

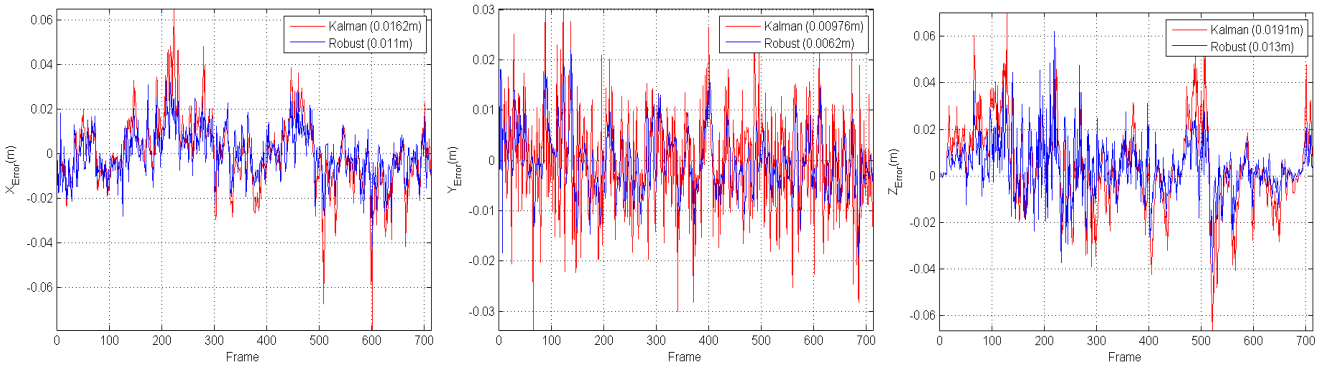


(a) X RMS (m) (b) Y RMSE (m) (c) Z RMSE (m)

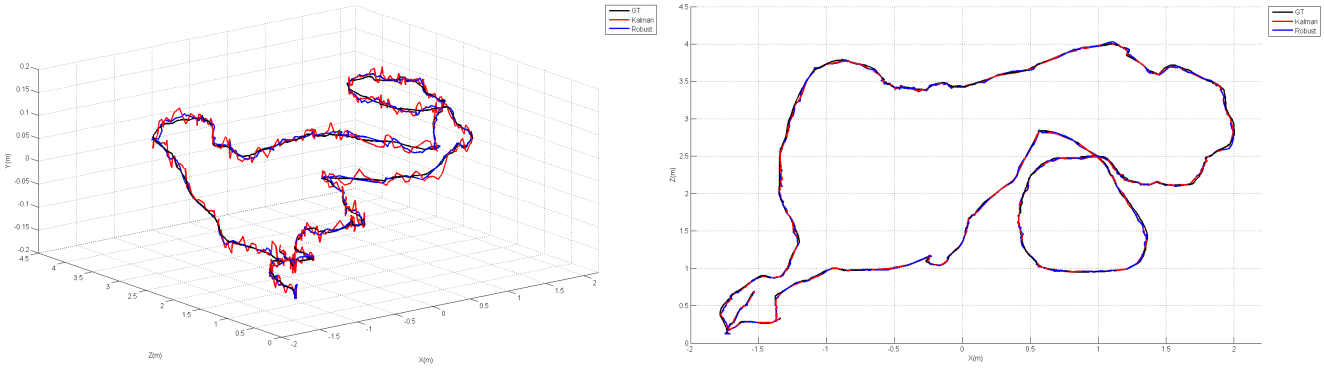


(d) 3D view of the trajectory (e) XZ view of the trajectory

Figure 16. $\boldsymbol{P_n}$ and $\boldsymbol{K_n}$ 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

Figure 17. $\boldsymbol{P_n}$, $\boldsymbol{K_n}$ and $\boldsymbol{Z_n}$ weighting results

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. 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 as the KF when the robot moves linearly according to the predefined motion model.

E. Orientation of the vehicle

To test the orientation angles delivered by our solution, we compute the error angle between the ground truth heading and

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 6. Final tracking error after CI filtering with $\boldsymbol{P_n}$ 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 7. Final tracking error after CI filtering with $\boldsymbol{P_n}$, $\boldsymbol{K_n}$ 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

TABLE 8. Final tracking error after CI filtering with $\boldsymbol{P_n}$, $\boldsymbol{K_n}$ and $\boldsymbol{Z_n}$ weighting

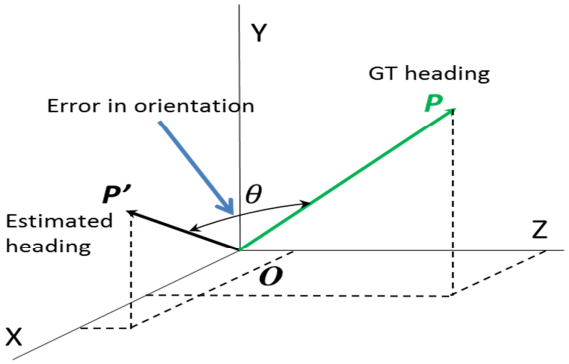


Figure 18. Angle between the GT and the estimated heading

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 Figure 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 equation (50):

$$P * P' = |P||P'| * cos(\theta) \tag{50}$$

From equation (50) we get:

$$cos(\theta) = \frac{P*P'}{|P||P'|} \tag{51}$$

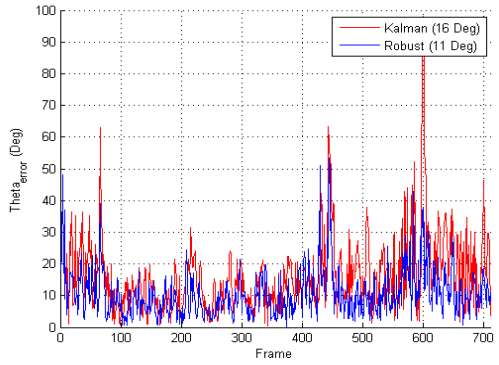
The error angle between the two directions becomes:

$$\theta = arcos\left(\frac{P*P'}{|P||P'|}\right) \tag{52}$$

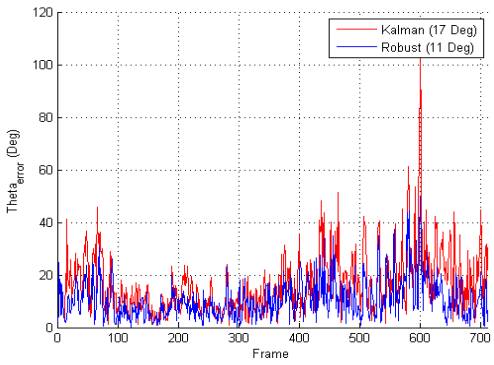
After applying equation (52), we got the results illustrated in Figure 19. The error in the orientation of the vehicle resulting from the RF was 11°. Whereas, KF-RMSE was 17°. With both filters the error in the orientation of the mobile robot was not important. More importantly, proportionally to the accuracy of the position of the vehicle, the application of CI better improved the accuracy of the orientation angle as follows: with $\boldsymbol{P_n}$ on its own RF-RMSE was 7.1° and KF-RMSE became 12°. With $\boldsymbol{P_n}, \boldsymbol{K_n}$ the results is even better where RF-RMSE became 4.8° and KF-RMSE 8.1°. Finally, the introduction of $\boldsymbol{Z_n}$ 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.

IX. Conclusion and Future Works

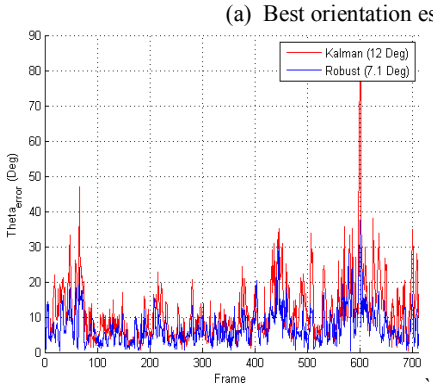
In this work, we presented a novel approach to accurately track moving vehicles in indoor environments with cooperative multiple consumer RGBD cameras. We described the details of our methodology and findings. Our findings about RGBD sensor correction are of importance for a more accurate measurement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



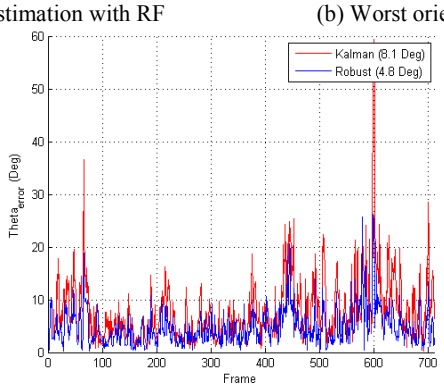
(a) Best orientation estimation with RF



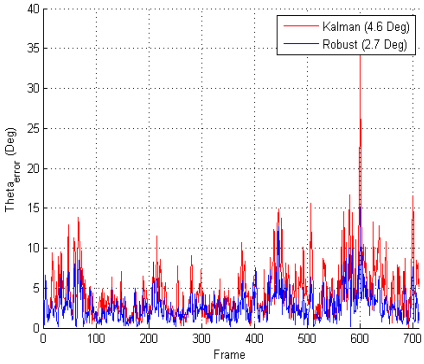
(b) Worst orientation estimation with RF



(c) CI with $\boldsymbol{P_n}$ weighting results



(d) CI with $\boldsymbol{P_n}$ and $\boldsymbol{K_n}$ weighting results



(e) CI with $\boldsymbol{P_n}$, $\boldsymbol{K_n}$ and $\boldsymbol{Z_n}$ weighting

Figure 19. Error in the estimated orientation of the vehicle

latter to overcome the lack of knowledge about the system governing the behaviour of the vehicles. We considered the quality of measurements and the estimates provided by each sensor in a CI algorithm. We successfully combined all the single contributions of the cameras in one consistent and cooperative output. Test results show the performance we achieved at a frame rate of 25Fps with the 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 the robust filtering and the covariance intersection algorithms, no parallelisation was required because they are just applied on some position data (five 3D points). Their linear computational nature requires a powerful sequential processing, which is the asset of the CPUs (their frequency of processing is much higher than the GPU's).

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 the 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*, vol. 74, no. 18, pp. 3823–3831, Nov. 2011.

[2] K. Litomisky, "Consumer RGB-D Cameras and their Applications," 2012.

[3] J. Fung and S. Mann, "OpenVIDIA," in *Proceedings of the 13th annual ACM international conference on Multimedia - MULTIMEDIA '05*, 2005, p. 849.

[4] A. Amamra and N. Aouf, "Real-Time Robust Tracking with Commodity RGBD Camera," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, 2013, pp. 2408–2413.

[5] A. Yilmaz, O. Javed, and M. Shah, "Object tracking," *ACM Comput. Surv.*, vol. 38, no. 4, p. 13–es, Dec. 2006.

[6] M. Taj and A. Cavallaro, "Multi-view multi-object detection and tracking," *Comput. Vis.*, 2010.

[7] R. Nevatia, "Self-calibration of a camera from video of a walking human," in *Object recognition supported by user interaction for service robots*, 2002, vol. 1, pp. 562–567.

[8] K. Okuma, A. Taleghani, and N. De Freitas, "A boosted particle filter: Multitarget detection and tracking," *Comput. Vision-ECCV*, 2004.

[9] S. Z. Li, "Multi-pedestrian detection in crowded scenes: A global view," in *2012 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.*, vol. 8, no. 4, pp. 369–393, Aug. 1999.

[11] S. Intille, J. Davis, and A. Bobick, "Real-time closed-world tracking," in *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.*, vol. 19, no. 7, pp. 780–785, Jul. 1997.

[13] J. Flusser and T. Suk, "Rotation Moment Invariants for Recognition of Symmetric Objects," *IEEE Trans. Image Process.*, vol. 15, no. 12, pp. 3784–3790, Dec. 2006.

[14] S. Y. Chen, "Kalman Filter for Robot Vision: A Survey," *IEEE Trans. Ind. Electron.*, vol. 59, no. 11, pp. 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," in *2011 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," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001, vol. 2, pp. II–786–II–793.

[17] M. Pupilli and A. Calway, "Real-Time Camera Tracking Using a Particle Filter.," *BMVC*, 2005.

[18] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience, 2006, p. 552.

[19] W. Niehsen, "Information fusion based on fast covariance intersection filtering," in *Proceedings of the Fifth International Conference on Information Fusion. FUSION 2002. (IEEE Cat.No.02EX5997)*, 2002, vol. 2, pp. 901–904.

[20] D. Smith and S. Singh, "Approaches to Multisensor Data Fusion in Target Tracking: A Survey," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 12, pp. 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*, vol. 56, no. 1, p. 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," in *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.*, vol. 31, no. 5, pp. 647–663, Feb. 2012.

[24] T. Nakamura, "Real-time 3-D object tracking using Kinect sensor," in *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, "KinectFusion," in *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, 2011, p. 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.*, vol. 18, no. 4, pp. 643–50, Apr. 2012.

[27] J. Han, L. Shao, D. Xu, and J. Shotton, "Enhanced computer vision with Microsoft Kinect sensor: a review.," *IEEE Trans. Cybern.*, vol. 43, no. 5, pp. 1318–34, Oct. 2013.

[28] "kinect_calibration/technical - ROS Wiki." [Online]. Available: http://wiki.ros.org/kinect_calibration/technical. [Accessed: 27-Jan-2014].

[29] R. Reyes, I. Lopez, J. J. Fumero, and F. de Sande, "accULL: An User-directed Approach to Heterogeneous Programming," in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, 2012, pp. 654–661.

[30] M. Andersen, T. Jensen, and P. Lisouski, "Kinect depth sensor evaluation for computer vision applications," 2012.

[31] M. Sedláček, "Evaluation of RGB and HSV Models in Human Faces Detection. Central European Seminar on Computer Graphics, Budmerice."

[32] Y. S. Hung and F. Yang, "Robust H∞ filtering with error variance constraints for discrete time-varying systems with uncertainty," 2003.

[33] D. Franken and A. Hupper, "Improved fast covariance intersection for distributed data fusion," in *2005 7th International Conference on Information Fusion*, 2005, vol. 1, p. 7 pp.

[34] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of Kinect depth data for indoor mapping applications.," *Sensors (Basel).*, vol. 12, no. 2, pp. 1437–54, Jan. 2012.

[35] A. Amamra and N. Aouf, "Robust and Sparse RGBD Data Registration of Scene Views," in *2013 17th International Conference on Information Visualisation*, 2013, pp. 488–493.

[36] C. G. M. P. A. P. S. S. Rita Cucchiara, "Improving Shadow Suppression in Moving Object Detection with HSV Color Information."

[37] M. S. Mahmoud, "Resilient linear filtering of uncertain systems," 2004.

[38] L. Xie, L. Lu, D. Zhang, and H. Zhang, "Improved robust H2 and H∞ filtering for uncertain discrete-time systems," 2004.

[39] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey.," *Cogn. Process.*, vol. 12, no. 4, pp. 319–40, Nov. 2011.

[40] J. Fung and S. Mann, “Using graphics devices in reverse: GPU-based Image Processing and Computer Vision,” in *2008 IEEE International Conference on Multimedia and Expo*, 2008, pp. 9–12.
[41] F. Jargstorff, “GPU Image Processing,” *SIGGRAPH 2004*, 2004.
[42] E. Kilgariff and R. Fernando, “The GeForce 6 series GPU architecture,” *ACM SIGGRAPH 2005 Courses*, 2005.