Intelligent Sensor-Scheduling for Multi-Kinect-Tracking

Florian Faion, Simon Friedberger, Antonio Zea, and Uwe D. Hanebeck

***Abstract*— This paper describes a method to intelligently schedule a network of multiple RGBD sensors in a Bayesian object tracking scenario, with special focus on Microsoft Kinect™ devices. These setups have issues such as the large amount of raw data generated by the sensors and interference caused by overlapping fields of view. The proposed algorithm addresses these issues by selecting and exclusively activating the sensor that yields the best measurement, as defined by a novel stochastic model that also considers hardware constraints and intrinsic parameters. In addition, as existing solutions to toggle the sensors were found to be insufficient, the development of a hardware module, especially designed for quick toggling and synchronization with the depth stream, is also discussed. The algorithm then is evaluated within the scope of a multi-Kinect object tracking scenario and compared to other scheduling strategies.**

I. Introduction

Building an RGBD-network with multiple Kinects requires addressing several issues. For example, as the sensors acquire the depth information using an active measurement principle, interference needs to be considered. Kinects detect depth information by using an infrared (IR) pattern projected onto the scene. However, when multiple devices are active at the same time, these patterns overlap with each other and cause interference and performance degradation. In addition, a major challenge is the large amount of raw data being collected. Assuming a standard VGA resolution of 640x480 pixels with 8 bits for color (before debayering) and 11 bits for depth, the bandwidth required for 30 fps is 21.8 MB/s of raw data per individual Kinect.

A. Contribution

The main contribution of this work is an algorithm to intelligently schedule the activation of devices from a Kinect network in a Bayesian tracking scenario. The objective is to minimize the expected uncertainty of the next measurement update. The presented method takes a stochastic sensor model for the Kinect depth sensor, the intrinsic parameters, and hardware constraints into account. This makes the algorithm more reliable and more exact than naïve approaches. For optimal IR-projector toggling, a new hardware module was developed that synchronizes to the depth data stream, marking an improvement upon the existing software and hardware solutions. A modified Kinect is depicted in Figure 1.

F. Faion, S. Friedberger, A. Zea, and U. Hanebeck are with the Institute for Anthropomatics, Intelligent Sensor-Actuator-Systems Laboratory (ISAS), Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany {faion|friedberger}@kit.edu, antonio.zea@student.kit.edu, uwe.hanebeck@ieee.org.

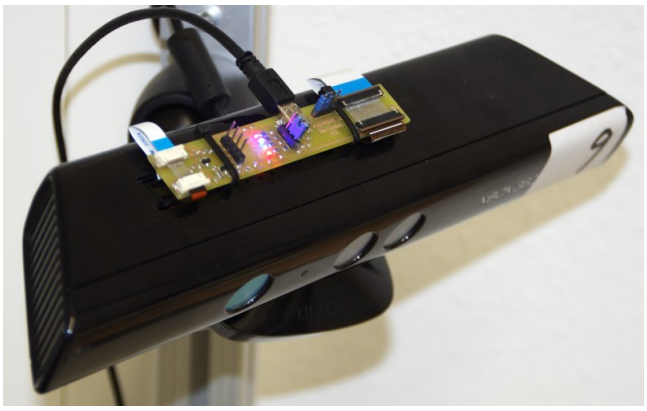


Fig. 1: Hardware modification for IR-projector-subsystem on/off-toggling of a Kinect device.

B. Related Work

Multiple Kinects have been used [1] to analyze a scene to create a so-called "parallel reality" art installation using a form of telepresence. The "LightSpace" setup [2], which tracks users using depth cameras and combines the result with multiple projectors to provide an interactive environment, also uses a similiar approach. In neither case [1], [2] were the interference of the depth sensors considered. Alternative hardware solutions have been considered, for example a mechanical shutter [3], in the context of dealing with interferences. However, the lack of synchronization between the shutter and the Kinect, among other factors, was shown to be detrimental in other applications, like markerless motion capture [4]. Work has also been done to solve the issue of interference [5] by interpolating missing values at the cost of additional computation time. Several libraries [6], [7] have been developed to work with the class of point clouds that are generated by depth cameras. Algorithms have been developed to track a sphere with dynamic position using Bayesian principles [8]. In a similar way to this paper, information theoretic principles have been used for sensor scheduling [9]. Additional work has been done analyzing the accuracy of the Kinect depth data [10], on which the sensor model presented in this work is based.

C. Overview

The remainder of this paper is structured as follows. In Section II, the considered problem is described in more theoretical and practical detail within the scope of an example tracking application. A stochastic motivation for the intelligent sensor selection is given in Section III. In Section IV, the developed algorithm is explained, followed by a descrip-

tion of the sensor model in Section V and the presentation of the toggling device in Section VI. The experimental results are given in Section VII. This includes a comparison with other scheduling approaches using as example the mentioned tracking scenario. Section VIII concludes this paper with a summary of the authors' insights and proposals for future work.

II. Problem Formulation

In this work, the problem of scheduling a network of multiple RGBD sensors in a Bayesian object tracking scenario is considered. The tracking scenario is assumed to contain n statically mounted RGBD Sensors $S = \{s_1, s_2, \ldots, s_n\}$ with given intrinsic $\{\mathbf{K}_1, \mathbf{K}_2, \ldots, \mathbf{K}_n\}$ and extrinsic $\{\mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_n\}$ calibrations. The vector $\underline{\boldsymbol{x}}_k$ describes the state parameters of a moving object at a given time step k. An overview of an example tracking scenario is depicted in Figure 2.

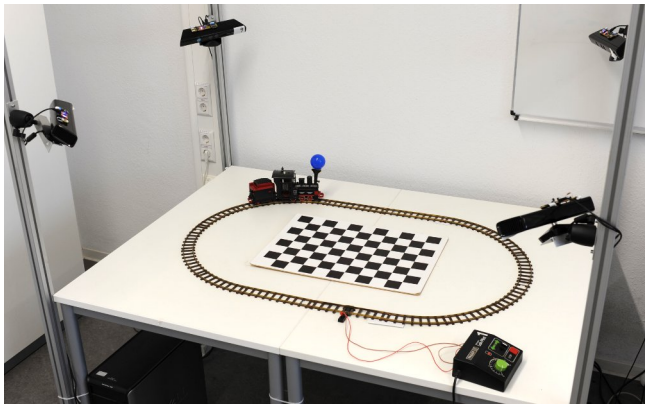


Fig. 2: Toy train tracking scenario with four Kinects.

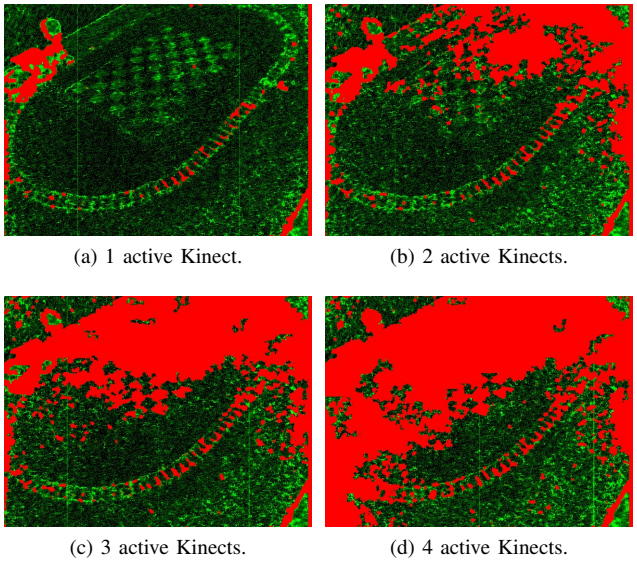
The goal is to measure the object using the *best* sensor $s_m \in S$ at each time step k, using time-multiplexing as the underlying schedule strategy. The advantages of this setup are:

1) a dramatic reduction of computational complexity and bandwidth as only one measurement has to be processed, and
2) avoidance of Kinect interference issues caused by overlapping fields of view, as shown in Figure 3.

The proposed setup involves a number of issues that need to be addressed. First of all, determining the *best* sensor requires the definition of a sensor quality measure. Second, after calculating the quality for each sensor at a certain time step k, the object could have moved farther, raising the need for some kind of prediction of future behavior. A further technical issue is that Kinects on their own do not allow rapid on/off-toggling of the IR-projector-subsystem.

III. Stochastic background

Tracking can be realized by means of a Bayesian state estimator, recursively estimating the state parameters $\underline{\boldsymbol{x}}_k$ of



(a) 1 active Kinect. (b) 2 active Kinects.

(c) 3 active Kinects. (d) 4 active Kinects.

Fig. 3: Illustration of raw shift variances, measured over 64 frames for a different number of active Kinects. Green pixels with varying brightness indicate valid values: *black* for 0 and *green* for 1. *Red* indicates at least one invalid value.

the object, where $\underline{\boldsymbol{x}}_k^e$ is a random vector[1] with estimated probability density $f_{\underline{\boldsymbol{x}}_k^e}(\underline{x})$. Bayesian tracking by design includes a time update, predicting the estimated state at a future time step $k+1$, and a measurement update, correcting a predicted state using a sensor measurement $\underline{\boldsymbol{y}}_{k+1}$. In consequence, improving quality is equivalent to minimizing the uncertainty of $\underline{\boldsymbol{x}}_{k+1}^e$. In this work, all uncertainties are assumed to be Gaussian, i.e., $\underline{\boldsymbol{x}}_k \sim \mathcal{N}(\hat{\underline{x}}_k, \mathbf{C}_{x_k})$ and $\underline{\boldsymbol{y}}_k \sim \mathcal{N}(\hat{\underline{y}}_k, \mathbf{C}_{y_k})$.

Given a probability density f from the space of Gaussian distributions $\mathcal{P}$ with $f \in \mathcal{P}$, a function

$$\mathcal{G} : \mathcal{P} \to \mathbb{R} \tag{1}$$

is required to measure its uncertainty. Since the target estimate is modeled as a Gaussian distribution, the determinant of the covariance matrix was chosen as an appropriate measure $\mathcal{G}$ with

$$\mathcal{G}\left(f(\underline{x}, \hat{\underline{x}}, \mathbf{C}_x)\right) = \det(\mathbf{C}_x) \ , \tag{2}$$

The goal is then to select the Kinect whose measurement will lead to the smallest expected uncertainty. Let m_s be the set of possible measurements for a sensor $s \in S$, then the best sensor is given by

$$s_m = \arg\min_{s \in S} \mathrm{E}_{\underline{y}_{k+1} \in m_s} \{\mathcal{G}(f(\underline{x}, \hat{\underline{x}}_{k+1}^e, \mathbf{C}_{x_{k+1}^e}))\} , \tag{3}$$

where $\mathrm{E}_{\underline{y}_{k+1} \in m_s}$ is the expected value over the measurements.

[1] We further denote predicted state parameters by $\underline{\boldsymbol{x}}^p$ and estimated (after measurement update) state parameters by $\underline{\boldsymbol{x}}^e$.

IV. Scheduling Algorithm

The developed sensor-scheduling algorithm will now be explained in detail. As mentioned before, a Bayesian state estimator is assumed for estimating the objects state parameters $\underline{\boldsymbol{x}}_k \sim \mathcal{N}(\underline{\hat{x}}_k, \mathbf{C}_{x_k})$.

A. Sensor Selection

The task of the tracking framework is to allow the sensor that yields the *expected best* measurement to perform the *real* measurement. More formally, the current estimate $\underline{\boldsymbol{x}}^e_k$ at time step k will be processed to select the sensor $s \in S$ which minimizes the covariance matrix $\mathbf{C}_{x^e_{k+1}}$ of $\underline{\boldsymbol{x}}^e_{k+1}$, where $k+1$ is the expected measurement time step. The entire sensor selection scheme is depicted in Algorithm 1 in pseudocode.

Algorithm 1 Sensor Selection

Input: estimate $\underline{\boldsymbol{x}}^e_k$, sensors S
1: $\underline{\boldsymbol{x}}^p_{k+1}$ = predict $\underline{\boldsymbol{x}}^e_k$
2: $\{\underline{\tilde{x}}_1, \ldots, \underline{\tilde{x}}_m\}$ = draw m samples from $\underline{\boldsymbol{x}}^p_{k+1}$
3: **for** $s \in S$ **do**
4: $\quad u_s = 0$
5: $\quad$ **for** $\underline{\tilde{x}} \in \{\underline{\tilde{x}}_1, \ldots, \underline{\tilde{x}}_m\}$ **do**
6: $\quad\quad \{\underline{\tilde{y}}_1, \ldots, \underline{\tilde{y}}_n\}$ = measure n times $\underline{\tilde{x}}$ with s
7: $\quad\quad$ **for** $\underline{\tilde{y}} \in \{\underline{\tilde{y}}_1, \ldots, \underline{\tilde{y}}_n\}$ **do**
8: $\quad\quad\quad \underline{\tilde{\boldsymbol{x}}}^e_{k+1}$ = update $\underline{\boldsymbol{x}}^p_{k+1}$ with $\underline{\tilde{y}}$
9: $\quad\quad\quad u_s$**+=** determinant of covariance of $\underline{\tilde{\boldsymbol{x}}}^e_{k+1}$
10: $\quad\quad$ **end for**
11: $\quad$ **end for**
12: $\quad u_s = \frac{u_s}{m \cdot n}$
13: **end for**
Output: $s_m = \arg\min_{s \in S} u_s$

First (see Line 1), by predicting the current estimate $\underline{\boldsymbol{x}}^e_k$ for the next expected measurement time step $k+1$, the value for $\underline{\boldsymbol{x}}^p_{k+1}$ is obtained. The next step is simulating measurements of this predicted state $\underline{\boldsymbol{x}}^p_{k+1}$ with each sensor $s \in S$ by means of a stochastic sensor model. Such a model calculates a noisy observation $\underline{\boldsymbol{y}} \sim \mathcal{N}(\underline{\hat{y}}, \mathbf{C}_y)$ for a given 3D point $\underline{\hat{y}}$. An appropriate sensor model for the Kinect is derived in Section V. Unfortunately, the Kinect sensor model cannot be directly applied to $\underline{\boldsymbol{x}}^p_{k+1}$ due to its nonlinearity. Instead, an approximation $\underline{\boldsymbol{x}}^p_{k+1}$ with m randomly drawn samples $\{\underline{\tilde{x}}_1, \ldots, \underline{\tilde{x}}_m\}$ (Line 2) is calculated.

For each sample $\underline{\tilde{x}}$, n measurements $\{\underline{\tilde{\boldsymbol{y}}}_1, \ldots, \underline{\tilde{\boldsymbol{y}}}_n\}$ are simulated with the stochastic sensor model for each sensor $s \in S$ (Line 5), because one measurement would not be able to capture the probability distribution when the object is close to the edge of the sensor field-of-view. Based on these n *virtual* measurements $\underline{\tilde{\boldsymbol{y}}}$ for each of the m sample states $\underline{\tilde{x}}$, measurement updates are performed (Line 7). Altogether we now have $m \cdot n$ expected estimations $\underline{\tilde{\boldsymbol{x}}}^e_{k+1} \sim \mathcal{N}(\underline{\tilde{x}}^e_{k+1}, \mathbf{C}_{\tilde{x}^e_{k+1}})$ for each sensor $s \in S$.

As explained in Section III, the determinant of the covariance matrix $\mathbf{C}_{\tilde{x}^e_{k+1}}$ is used as a measure for the uncertainty of $\underline{\tilde{\boldsymbol{x}}}^e_{k+1}$. It is averaged over all m and n and the result is one scalar value

$$u_s = \frac{1}{m \cdot n} \sum_{m \cdot n} \det(\mathbf{C}_{\tilde{x}^e_{k+1}}), \tag{4}$$

representing the state uncertainty in time step $k+1$ for the current sensor. Equation 4 is analogous to Line 9 and 12. Finally, after calculating the uncertainty u_s for each sensor $s \in S$, the minimum u_s (Line 13) characterizes the *best* sensor s_m. This completes the sensor-scheduling and the *best* sensor is activated to perform a real measurement $\underline{\boldsymbol{y}}_{k+1}$. This measurement is then used to determine the estimation $\underline{\boldsymbol{x}}^e_{k+1}$ for the next time step $k+1$.

V. Stochastic Sensor Model for Kinect

In this section, a stochastic sensor model for the Kinect depth sensor is derived. This model calculates a Gaussian distribution $\underline{\boldsymbol{y}} \sim \mathcal{N}(\underline{\hat{y}}, \mathbf{C}_y)$ for a given 3D point $\underline{\hat{y}}$ that is visible to the sensor. Based on this distribution, noisy observations of this point can be produced.

A. Visibility Constraints

Similar to standard RGB cameras, the view region of a Kinect depth sensor can be modeled as a pyramidal frustum, whose parameters can be derived from the intrinsic calibration $\mathbf{K}$ and distance constraints inherent to the hardware [11]. These constraints are specified as $[0.8\,\text{m}, 3.5\,\text{m}]$. A scheme of the viewable area is depicted in Figure 4.

B. Stochastic Model

Kinects do not deliver depth values directly. Instead, the values received are what in the PrimeSense code [12] is described as *shift values*, which are integer 11-bit values representing an encoded form of the triangulation disparities. The relationship between *shift values* and disparities is assumed to be linear [7].

It is assumed that no unexpected sources of noise, e.g., additional Kinects, are present. For a given point $\underline{\hat{y}}$, let s_{max} and s_{min} be the extrema of the measured shifts. These shifts can either be invalid (greater than a given threshold, e.g., $s_{thresh} \approx 1023$ for 5 m), on edges (in which case one can observe that $s_{max} - s_{min} > 2$), or are stable (in which case usually $s_{max} - s_{min} = 1$). This means that shift values for stable depths generally vary discretely from one value to directly adjacent ones, and the influence of angle, the depth in meters, reflectivity or other factors is not as great as expected. It should be noted that the noise variance of measurements is not constant in time, not even in static scenes, as artifacts in the algorithm (such as vertical bands appearing in certain frames, among others) cause additional instability. Some of these effects can be seen in Figure 3.

For the noise model the shift is modelled as $\boldsymbol{s} \sim \mathcal{N}(\hat{s}, \sigma_s^2)$, using an upper value of $\sigma_s^2 = \frac{1}{3}$. Actual variances are generally smaller, and this value is only considered as an upper threshold. Measurements with smaller variances can be assumed to be stable, while those with greater variances are usually on edges.

There are a few formulas published to convert the shift value to a depth value in meters. A formula provided in [13], shown to be very close to empirical results, is:

$$z(s) = a \cdot tan(\frac{s}{b} + c) \ , \tag{5}$$

with $a = 0.1236$, $b = 2842.5$, $c = 1.1863$, and z as the distance in meters. Observing the transform one can see that the function between shift to depth is relatively flat in the interval $[\hat{s} - 1, \hat{s} + 1]$ for any given $\hat{s}$ in the depth interval from 0.8 m to 3.5 m. This simplifies the transformation of the shift distribution through linearization in the neighborhood of $\hat{s}$

$$z(s) \approx z'(\hat{s}) \cdot (s - \hat{s}) + k \ , \tag{6}$$

for a constant k, which leads to

$$\sigma_z^2 \approx (z'(\hat{s}))^2 \cdot \sigma_s^2 = \left(\frac{a}{b} \sec^2(\frac{\hat{s}}{b} + c)\right)^2 \cdot \sigma_s^2 \ . \tag{7}$$

Let u and v be the pixel coordinates of $\underline{y}$. In order to unproject a given point $(u, v, s)^T$ to 3D-coordinates, it is also useful to consider noise values for u and v. Given how little is known about the PrimeSense triangulation algorithm, it is hard to say authoritatively how certain the pixel coordinates are. With this in mind, it is reasonable to model u and v as $\boldsymbol{u} \sim \mathcal{N}(\hat{u}, \sigma_u^2)$ and $\boldsymbol{v} \sim \mathcal{N}(\hat{v}, \sigma_v^2)$ with $\sigma_u^2 = \sigma_v^2 = \frac{1}{3}$ as upper estimations.

The transformation from 2D to 3D-coordinates is assumed to obey the pinhole principle. Assuming

$$\mathbf{K} = \begin{pmatrix} f_u & 0 & c_u \\ 0 & f_v & c_v \end{pmatrix} \tag{8}$$

as the intrinsic calibration, it follows that

$$\underline{y} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \frac{u - c_u}{f_u} \\ \frac{v - c_v}{f_v} \\ 1 \end{pmatrix} z(s) \tag{9}$$

holds. Based on these equations, the covariance matrix of the 3D-coordinates can be calculated as

$$Var(x_1) = \frac{(\hat{u} - c_u)^2 \sigma_z^2 + z^2 \sigma_u^2 + \sigma_z^2 \sigma_u^2}{f_u^2} \, , \tag{10}$$

$$Var(x_2) = \frac{(\hat{v} - c_v)^2 \sigma_z^2 + z^2 \sigma_v^2 + \sigma_z^2 \sigma_v^2}{f_v^2} \, , \tag{11}$$

$$Var(x_3) = \sigma_z^2 \, , \tag{12}$$

$$Cov(x_1, x_3) = \frac{\hat{u} - c_u}{f_u} \sigma_z^2 \, , \tag{13}$$

$$Cov(x_2, x_3) = \frac{\hat{v} - c_v}{f_v} \sigma_z^2 \, , \tag{14}$$

$$Cov(x_1, x_2) = \frac{\hat{u} - c_u}{f_u} \cdot \frac{\hat{v} - c_v}{f_v} \sigma_z^2 \, . \tag{15}$$

The sensor model for a given point $\hat{\underline{y}}$ leads to the noisy observation $\underline{\boldsymbol{y}} \sim \mathcal{N}(\hat{\underline{y}}, \mathbf{C}_y)$, with

$$\mathbf{C}_y = \begin{pmatrix} Var(x_1) & Cov(x_1, x_2) & Cov(x_1, x_3) \\ Cov(x_1, x_2) & Var(x_2) & Cov(x_2, x_3) \\ Cov(x_1, x_3) & Cov(x_2, x_3) & Var(x_3) \end{pmatrix} . \tag{16}$$

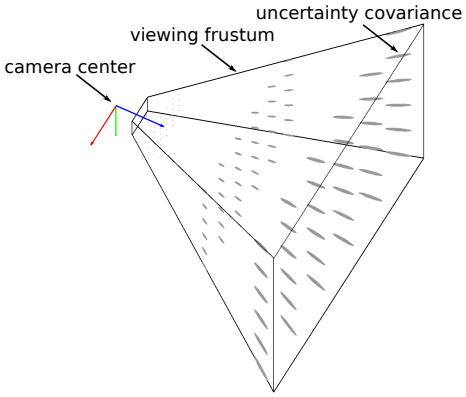


Fig. 4: Scheme of the stochastic Kinect sensor model. For some selected locations the covariance ellipsoids are qualitatively depicted in gray.

VI. Toggling Mechanism

A sensor setup using multiple Kinects with overlapping views will produce interference artifacts due to overlapping IR patterns, as explained in Section II. This raises the need for time-multiplexing, where a single Kinect is chosen to be active at any given time while the other Kinects are powered down. In this section, two solutions are discussed for toggling the Kinect IR-projector-subsystem: one in software and one in hardware.

A. Software Toggling

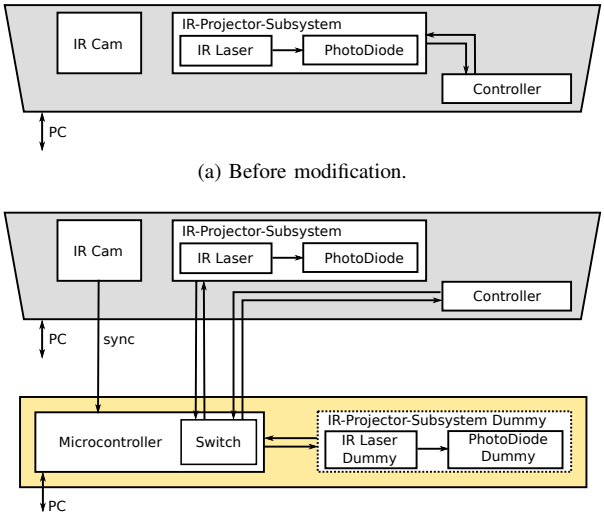
Software toggling uses the driver functions and is the first obvious idea to control the IR-projector-subsystem. However, the software solutions available within the standard frameworks libfreenect [13], OpenNI [14], and KinectSDK [15] were found to be insufficient. While toggling of the IR-projector-subsystem is indeed supported by these drivers, it turned out that this mechanism takes up to 200 ms from turning on the projector to obtaining a stable depth image, causing the loss of at least six frames. Quick toggling was also observed to cause instability, in a few cases even causing the driver to cease to respond. This problem makes software toggling impractical for tracking applications, especially when fast switching between Kinects is needed.

B. Hardware Toggling

A hardware solution was designed to solve the toggling issues. The IR-projector-subsystem before modification is depicted in Figure 5a. The main component of the projector is a laser diode projecting the IR pattern. In an unmodified device, the brightness of the projector is constantly monitored by a photodiode and then transmitted to a controller that regulates the current of the laser diode. If the IR-projector subsystem is directly turned off, the controller will stop receiving the signal and this will cause the Kinect to crash inevitably.

To solve this, a subsystem (in the following denoted as the *dummy IR-projector-subsystem*) was developed to emulate

the IR-projector-subsystem for the controller. A microcontroller acts as an electronic switch to toggle between the two systems. A schematic of the modified system is illustrated in Figure 5b. Furthermore, to avoid incomplete depth images caused by turning off the projector while the IR camera is capturing a frame, a synchronization routine to the toggling mechanism was added which adjusts the *frame valid* line of the IR camera. The output is set to 1 if the current frame is completely captured, else it is set to 0. The electronic switch was then only triggered on a *frame valid* event, which in turn confidently leads to complete depth images. This modification is also illustrated in Figure 5b.



(a) Before modification.

(b) After modification.

Fig. 5: Schematic illustration of the toggling module.

Technical Realization: A top and bottom view of the board that implements the hardware toggling is depicted in Figure 6. An MSP430 micro controller from Texas Instruments is used as the core component (1). The board is connected to the PC via a USB interface (2). This interface is also used as the power supply of the board. The FFC conducting path from the IR-projector-subsystem to the controller is redirected to the board (3). The IR-projector-subsystem emulation required both the photodiode and the laser to be simulated. The photodiode dummy consists merely of constant voltage, implemented by diode (4a) and a resistor (4b), while the running laser was emulated using the dummy current of a resistor (5). For synchronization the FFC conducting path from the IR camera to the Kinect is redirected to the micro controller (6). A modified Kinect is depicted in Figure 1.

VII. Evaluation

This section describes the evaluation of the intelligent scheduling algorithm using the tracking scenario depicted in Figure 2.

A. Setup

An RGBD camera network, consisting of four rigidly mounted Kinects, is observing a scenario containing a toy train tagged with a spherical blue landmark. An Unscented

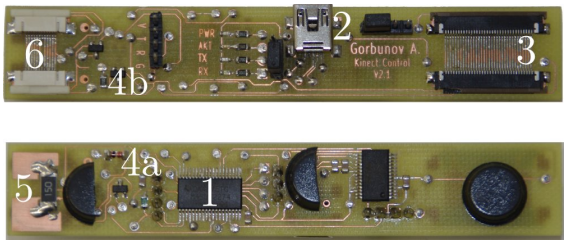


Fig. 6: Kinect toggling module.

Error	All sensors	Cycle sensors	Intelligent selection
Avg.	5.0 mm	9.8 mm	6.9 mm
Std. dev.	4.1 mm	13.9 mm	4.1 mm
Max.	27.0 mm	200.2 mm	21.2 mm

TABLE I: Comparison of average distances to track.

Kalman Filter [16] is used to track the sphere using the algorithm presented by [8]. The system behavior is assumed to evolve according to a constant velocity model.

B. Experiments

Because of the small size of the setup, the camera fields of view overlap in several places. As mentioned before, this causes intense interference. Several experiments with different sensor-scheduling algorithms were carried out and the resulting tracking accuracies were subsequently analyzed in detail. A brief description of the applied scheduling strategies follows.

All Sensors: This is the reference approach. Simultaneously measuring with all sensors is a naïve but popular scheduling strategy. Existing interferences between sensors are completely ignored. This allows far more measurements to be used, but many of the potential measurements are lost due to interference. This approach involves the highest computational effort as the raw data of every single sensor has to be processed.

Cycle Sensors: This strategy sequentially cycles through all sensors (with a period of 1 s) without consideration of whether the object is visible.

Intelligent Sensor Selection: This scheduling strategy makes use of the selection algorithm developed in Section IV that minimizes the uncertainty of the estimated object position. The optimal sensor for the next measurement is determined every 200 ms.

C. Results

The ground truth was determined by carefully measuring and modelling the real world track. Then, for the first 500 time steps, the closest point on the track corresponding to the measurement was calculated and its distance measured. The resulting errors are listed in Table I.

On one side, the results show that the naïve approach, even after considering the interference, produces a slightly smaller error. This may sound surprising, but it is merely a consequence of the small size of the scenario. In this case, the fusion of four measurements with more noise yields a smaller uncertainty than a single measurement with little

noise. Considering how quickly the noise grows in relation to depth, a theoretical alternative of increasing the amount of sensors to compensate for greater distances can be quickly seen as not viable.

Additionally, it must also be pointed out that using multiple sensors to compensate for the noise also causes a dramatic increase in bandwidth and extremely noisy depth images, which may be unusable for further applications. This further shows the benefits of using the other two approaches. Since only one sensor remains active at any given time, the bandwidth is constant and the quality of the depth image remains optimal.

Figures 7, 8, and 9 contain the results for one single round trip of the toy train. Figure 7, 8, and 9 (a) shows the estimates after a measurement update in the corresponding color of the measuring Kinect. Figure 7, 8, and 9 (b) shows the direction of the estimated velocity.

For the cyclic toggling approach, the visibility of the object is not guaranteed. This causes the corresponding measurements in some points of the track to be missing, as can be seen in Figure 8a.

Figure 9a shows that the best sensor, as chosen by the algorithm, usually defaults to the closest sensor to the measurement. This is quite intuitive and follows directly from the sensor noise model, which dictates that the uncertainty increases with the depth distance. This does not have to be the case, however, in cases when the cameras have different shift-to-depth or intrinsic parameters. Another exception to this rule happens when the object is outside the view frustrum of the closest camera.

VIII. Conclusion

In this work, a method was presented to intelligently schedule an RGBD sensor network for tracking applications. The proposed algorithm activates the sensor that yields the best measurement, as defined by a novel stochastic model that also considers hardware constraints and intrinsic parameters. This requires quick toggling and synchronization with the depth stream, for which a specialized hardware module was developed.

The evaluation compared the performance of the proposed intelligent scheduling algorithm against naïve approaches such as having all sensors activated and cycling through all sensors. Results showed the intelligent algorithm performing in a similar way to the "all sensor" strategy, while using four times less samples, requiring much less bandwidth, and avoiding issues like interference due to multiple active sensors.

Furthermore, the presented method can be applied to any multi-sensor tracking system that can be described using stochastic sensor models.

Future Work

For simultaneous measurements with more than one sensor, interference needs to be considered by the sensor model. A horizon of future measurements, i.e., measurement series, may also be of interest. A more algorithmic improvement would be to deterministically approximate the predicted state. Finally, applying sensor-scheduling to extended objects or entire areas could be useful for applications like free viewpoint television [17] or telepresence [18].

IX. Acknowledgements

We would like to thank Alexander Riffel and Anton Gorbunov for implementing the toggling-hardware. Furthermore, we would like to thank Jannik Steinbring for implementing the tracking algorithm.

References

[1] T. Takeuchi, T. Nakashima, K. Nishimura, and M. Hirose, "Prima: parallel reality-based interactive motion area," in *ACM SIGGRAPH 2011 Posters*, ser. SIGGRAPH '11. New York, NY, USA: ACM, 2011, pp. 80:1–80:1.

[2] A. Wilson and H. Benko, "Combining multiple depth cameras and projectors for interactions on, above and between surfaces," in *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. ACM, 2010, pp. 273–282.

[3] A. Scholz and K. Berger, "Multiple Kinect Studies," *Computer*, 2011.

[4] K. Berger, K. Ruhl, Y. Schroeder, C. Bruemmer, A. Scholz, and M. Magnor, "Markerless Motion Capture using multiple Color-Depth Sensors," *Sensors (Peterborough, NH)*, 2011.

[5] A. Maimone and H. Fuchs, "Encumbrance-free telepresence system with real-time 3D capture and display using commodity depth cameras," *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pp. 137–146, Oct. 2011.

[6] R. R. B. Rusu, S. Cousins, and W. Garage, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[7] ROS.org, "Technical description of Kinect calibration," 2010. [Online]. Available: http://www.ros.org/wiki/kinect_calibration/technical

[8] M. Baum, V. Klumpp, and U. D. Hanebeck, "A Novel Bayesian Method for Fitting a Circle to Noisy Points," in *Proceedings of the 13th International Conference on Information Fusion (Fusion 2010)*, Edinburgh, United Kingdom, 2010.

[9] G. Hoffmann and C. Tomlin, "Mobile sensor network control using mutual information methods and particle filters," *Control, IEEE Transactions on*, vol. 55, no. 1, pp. 32–47, Jan. 2010.

[10] K. Khoshelham, "Accuracy analysis of kinect depth data," in *ISPRS Workshop Laser Scanning*, vol. 38, 2011.

[11] PrimeSense, "Kinect reference," 2010. [Online]. Available: https://github.com/adafruit/Kinect/

[12] ——, "PrimeSense/Sensor GitHub repository," 2012. [Online]. Available: Source/XnDDK/XnShiftToDepth.cpp

[13] "libfreenect Framework," 2010. [Online]. Available: http://openkinect.org

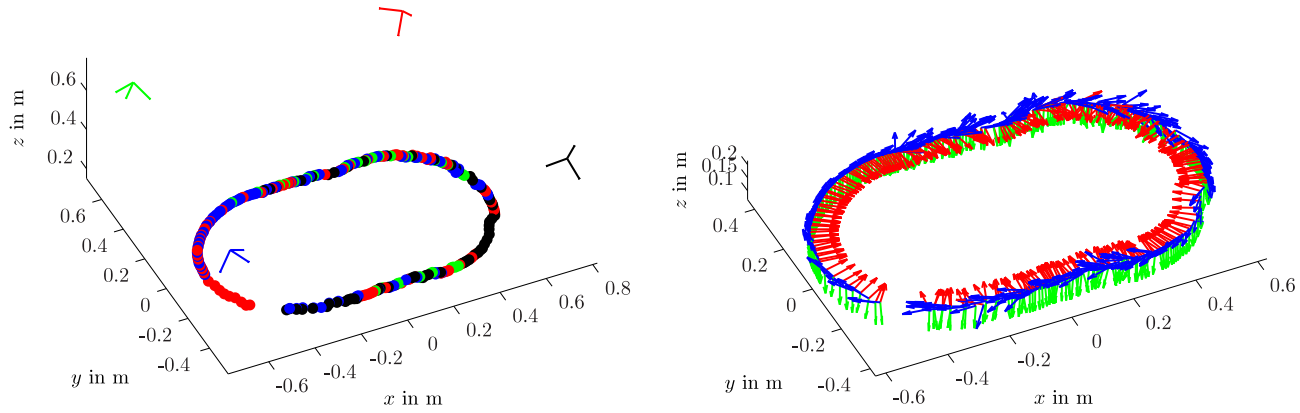
[14] the OpenNI organization, "OpenNI Framework," 2012. [Online]. Available: http://www.openni.org/

[15] Microsoft, "Kinect SDK," 2012. [Online]. Available: http://www.microsoft.com/en-us/kinectforwindows/

[16] S. J. Julier and J. K. Uhlmann, "Unscented Filtering and Nonlinear Estimation," *Computer Engineering*, vol. 92, no. 3, 2004.

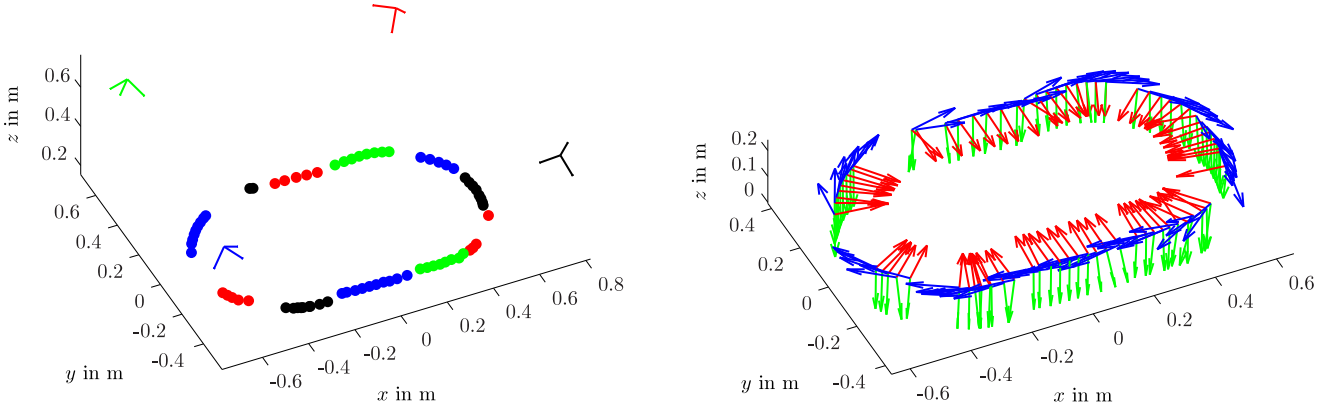
[17] M. Tanimoto, M. Tehrani, and T. Fujii, "Free-viewpoint TV," *Signal Processing*, no. January, pp. 67–76, 2011.

[18] F. Packi, A. P. Arias, F. Beutler, and U. D. Hanebeck, "A Wearable System for the Wireless Experience of Extended Range Telepresence," in *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, Taipei, Taiwan, 2010.



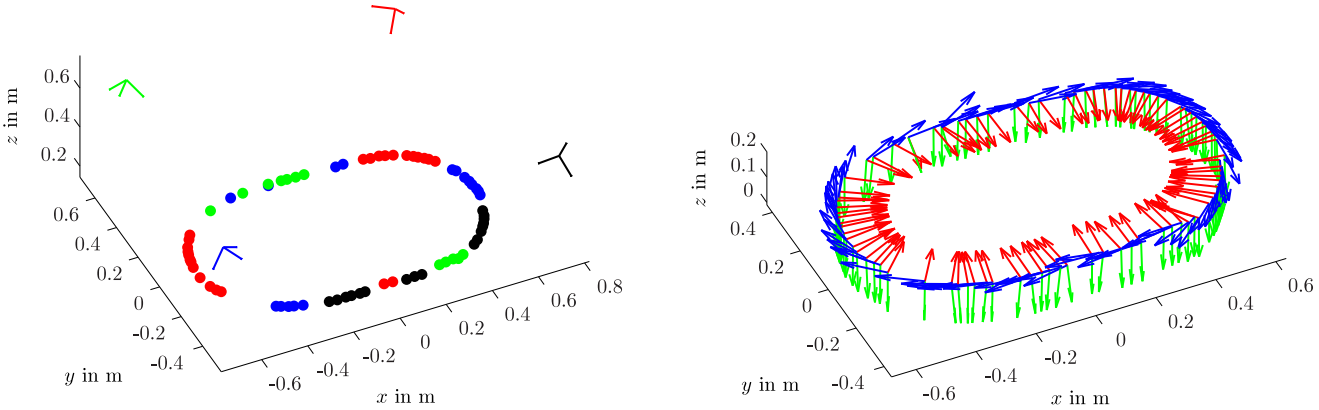
(a) Mean $\hat{\underline{x}}^e$ of updated estimates. (b) Direction of velocity.

Fig. 7: Results of scheduling algorithm *all sensors*.



(a) Mean $\hat{\underline{x}}^e$ of updated estimates. (b) Direction of velocity.

Fig. 8: Results of scheduling algorithm *cycle sensors*.



(a) Mean $\hat{\underline{x}}^e$ of updated estimates. (b) Direction of velocity.

Fig. 9: Results of scheduling algorithm *intelligent sensor selection*.