Real-Time Fine Grained Occupancy Estimation using Depth Sensors on ARM Embedded Platforms

Sirajum Munir[1], Ripudaman Singh Arora[2], Craig Hesling[3], Juncheng Li [4,], Jonathan Francis[5], Charles Shelton[6], Christopher Martin[7], Anthony Rowe[8], and Mario Berges[9]
Bosch Research and Technology Center, Pittsburgh, PA [1,4,5,6,7]
University of Michigan, Ann Arbor, MI[2]
Carnegie Mellon University, Pittsburgh, PA[3,4,8,9]
Email: {sirajum.munir[1], billy.li[4], jon.francis[5], charles.shelton[6], christopher.martin[7]}@us.bosch.com,
asripu@umich.edu[2], craig@hesling.com[3], agr@andrew.cmu.edu[8], marioberges@cmu.edu[9]

Abstract—**Occupancy estimation is an important primitive for a wide range of applications including building energy efficiency, safety, and security. In this paper, we explore the potential of using depth sensors to detect, estimate, identify, and track occupants in buildings. While depth sensors have been widely used for human detection and gesture recognition, computer vision algorithms are typically run on a powerful computer like XBOX or Intel® Core™ i7 processor. In this work, we develop a prototype system called FORK using off-the-shelf components that performs the entire depth data processing on a cheaper and low power ARM processor in real-time. As ARM processors are extremely weak in running computer vision algorithms, FORK is designed to detect humans and track them in a very efficient way by leveraging a novel lightweight model based approach instead of traditional approaches based on histogram of oriented gradients (HOG) features. Unlike other camera based approaches, FORK is much less privacy invasive (even if the sensor is compromised). Based on a complete implementation, real-world deployment, and extensive evaluation at realistic scenarios, we observe that FORK achieves over 99% accuracy in real-time (4-9 FPS) in occupancy estimation.**

Index Terms—**Occupancy Estimation, People Counting**

I. Introduction

Occupancy estimation is an important primitive for a wide range of applications including building energy efficiency, safety, and security. Heating, ventilation, and air conditioning (HVAC) is a major source of energy consumption in the US as approximately 35% of the total energy in the US was used for HVAC in 2006 [2]. Most HVAC systems operate by assuming maximum occupancy in each room, which leads to a significant waste of energy, e.g., an HVAC system providing ventilation for 30 people when there are only 10 people in a room [15]. Feeding occupancy count to HVAC systems enables reducing such energy waste and enables zone based heating and cooling control. When it comes to lighting control, PIR motion sensor based solutions often turn off lights when people are inside but not moving, and keep the lights on for a timeout interval even after everyone leaves a room, thus wasting energy. Estimating the number of occupants in real-time enables addressing both issues. In emergency situations, e.g., in a fire, an accurate occupancy estimation solution is very useful for rescue operations. Also, in banks, museums, kindergartens, and high schools, occupancy estimation helps to determine whether everyone has left at the end of the day to improve safety and security.

Several occupancy estimation solutions are available in the market and in the literature that use break-beam sensors, ultrasonic sensors, cameras, and thermal imagers. Break-beam sensor based solutions do not work when multiple people enter or leave through a door simultaneously. Ultrasonic sensor based solutions [25] require significant training and they are usually not pet friendly. RGB cameras are too privacy invasive to be deployed in many settings, e.g., in office rooms, and they do not work in the dark. Low resolution (e.g., an 8x8 pixel Panasonic GridEYE) thermal imagers do not have good enough resolution to detect multiple people entering/exiting simultaneously and high resolution thermal imagers are very expensive, e.g., a 32x32 thermopile array sensor costs over $200 (Heimann sensor[3]). Our exploratory study shows that a depth sensing based solution can overcome these limitations a great extent as our system can detect multiple people entering/leaving a room simultaneously, requires almost no training, is not as privacy invasive as RGB cameras, is pet friendly (IR signals don't bother pets), and works in a completely dark environment (because of the underlying properties of the IR structured light being used in the depth sensor). We also show how depth data has the promise for being a lightweight biometric primitive to identify room occupants for personalizing room environment and to track the flow of people through a building.

To explore these ideas, we build a prototype system using off-the-shelf components. The system is based on a Time of Flight (TOF) sensor, more specifically a Kinect depth sensor. We call the system FORK (Fine grained Occupancy estimatoR using Kinect). Although Kinect has been extensively used for human detection, skeletal tracking, and even gesture recognition, our solution is different from others for two reasons. First, most of the previous works assume that the Kinect is placed in front of people, which is not practical in a number of settings, e.g, in office rooms and classrooms. Our solution places the Kinect at the ceiling (Figure 1(a)) near to a door, which resolves this issue. This placement also helps us to deal with occlusion. Second, instead of using a powerful computer like XBOX, we tailor the solution to be able to run in a low power ARM processor-based embedded computer, e.g., ODROID-XU4 [6] (Figure 1(c)). ARM processors are extremely weak in running computer vision algorithms. As a micro benchmark, an occupancy estimation solution that typically runs at 25 FPS on an Intel® Core^{TM} i5 machine runs at 2 FPS when directly ported to an ARM processor machine. Low frame processing rate violates assumptions of existing tracking algorithms (described in Section II-E) and requires a novel solution with minimal computational complexity.

FORK uses a model based approach for occupancy estimation with several steps. First, after preprocessing, it performs multilevel scanning and extracts contours of potential heads. Using the contours, it finds the minimum enclosing circles of the contours, which provides approximate centers and radii of the heads. Second, for each identified circles, it uses the 3D depth data to verify whether it is an individual or not. To do so, it uses a novel orientation invariant 3D human model that relies on anthropometric properties of human heads and shoulders, as these are the most prominent body parts seen from the ceiling. Finally, FORK detects door locations and tracks individuals to determine whether they are entering or leaving a room for estimating number of occupants in a room. Note that FORK doesn't use the Kinect SDK and hence can generalize to any depth camera with similar hardware functionality.

This work has four major research contributions. First, we perform a comprehensive exploratory analysis to understand the potential of depth sensors for detecting, estimating, and tracking building occupants using low cost embedded processors. It includes the design and implementation of a novel lightweight solution for occupancy estimation, which is the first solution that runs on an ARM processor and does the processing in real-time. We also investigate how different factors affect the real-time performance of FORK. Second, we explore how the system can be used to identify and track occupants by extracting biometric features and using machine learning algorithms. Third, we design the system in a way that requires almost no training as FORK can determine door locations automatically from the depth data. Fourth, we have deployed nine instances of FORK at a Bosch office and several CMU classrooms. Based on evaluation at realistic scenarios (e.g., door opening and closing, people moving with bicycles, cleaning lady moving with a large drum of cleaning equipment, food caterers moving with boxes of food in hands, people carrying laptops, people wearing caps), we see that FORK achieves over 99% accuracy in real-time (4-9 FPS) in occupancy estimation.

II. Exploratory Analysis of Depth Sensing Based Approaches

In this section, we explore the potential of modern depth sensors to detect occupancy patterns in buildings on ARM embedded platforms. In particular, we focus on solutions for human detection, counting, tracking, and identification as well as the detection of other non-human objects of interest (e.g., doors). Our experiments were carried out using our prototype system, FORK. We begin the section by describing FORK.

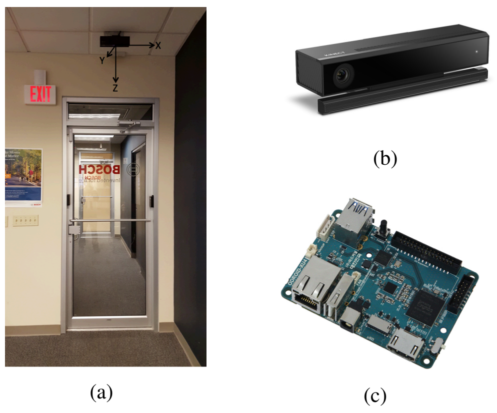

Fig. 1. (a) Placement of a Kinect sensor on a ceiling. (b) Kinect sensor for XBOX One. (c) Embedded computer Odroid-XU4.

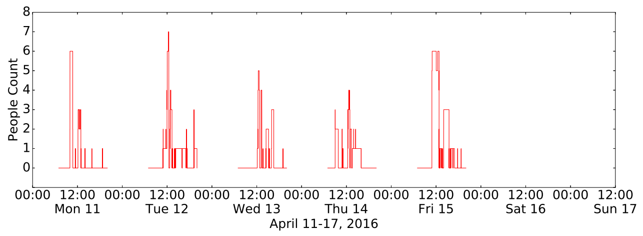

Fig. 2. Occupancy estimation for one week

A. FORK Overview

We use the depth sensor in Kinect for XBOX One (Figure 1(b)) in this work. We choose Odroid-XU4 (Figure 1(c)) for processing as it is one of the few embedded platforms that supports USB 3.0 that is required by this Kinect. We upload the occupancy count over WiFi to the Sensor Andrew [22] infrastructure, which allows monitoring of occupancy patterns in real-time using a browser by authorized users. As an example, Figure 2 shows how the occupancy pattern of a conference room (Warhol) of a Bosch office changes over a typical workweek. FORK does not store or upload any images for privacy concerns.

An overview of FORK occupancy estimation approach is shown in Figure 3. After doing minimal preprocessing, FORK performs multilevel scanning, where it scans at a few potential depth levels to detect humans. For each level, FORK extracts contours of potential heads by ignoring the depth data below that level. Then it verifies whether each contour represents a real person by verifying the presence of a head and a shoulder using anthropometric properties of a human body. FORK tracks individuals going through a door to count number of people inside. It also determines the location of the door.

FORK is implemented in C++. It uses OpenCV library and runs on Ubuntu 15.04 as an application on Odroid XU4. To access the depth frames from a Kinect, FORK uses *libfreenect2* library of the OpenKinect project. *Libfreenect2* library leverages *libusb* library to access the USB interface. FORK runs several modules to perform several tasks to report

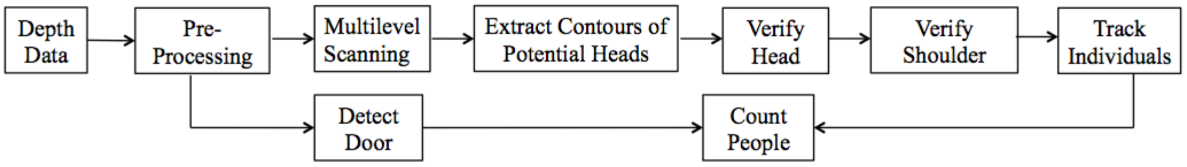

Fig. 3. An overview of the FORK approach

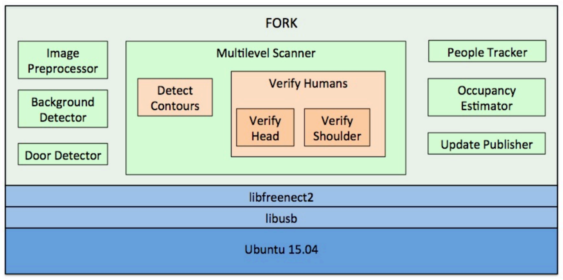

Fig. 4. FORK software architecture

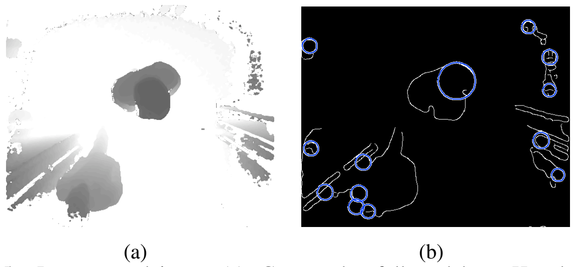

Fig. 5. Preprocessed image (a). Canny edge followed by a Hough circle transformation (b).

occupancy estimation information at real-time. These modules and FORK software architecture are shown in Figure 4. The *Image Preprocessor* module preprocesses depth frames. The *Background Detector* module captures an approximate background. The *Door Detector* module detects the location of the door. *Multilevel Scanner* module is crucial for detecting and locating humans. It has a few sub-modules for contour detection and verifying the presence of heads and shoulders. Once the presence of someone is verified, the *People Tracker* module tracks whether he is entering or exiting through the door and *Occupancy Estimator* module updates the people count. The *Update Publisher* module publishes the updated count to Sensor Andrew server [22] using XMPP communication protocol. The detailed approach is described below.

B. Preprocessing

Kinect for XBOX One produces 512x424 resolution depth images at 30 FPS. Each pixel of a depth frame provides the distance from the Kinect to the nearest objects in millimeters. However, in the presence of noise, the corresponding pixel has a value of 0. State of the art approaches [28] use median filtering to smooth the depth image. However, we find that a 5x5 window median filtering takes 3x more computation time than that of our entire occupancy estimation algorithm. Also, as most noise is at the perimeter of the frame, FORK doesn't use median filtering by default. However, it can be configured to do so. We evaluate FORK with and without median filtering (Section IV-C2 and IV-C1). At the preprocessing step, we reset noise pixels and outliers (depth too high) to floor depth. The floor depth is computed by computing a histogram of the depth values of a frame, where the bin with the maximum number of depth data points is considered the floor. A preprocessed image is shown in Figure 5(a), which shows that a portion of a head (left person) is missing due to noise.

C. Human Detection

FORK detects humans in three steps: multilevel scanning, head verification, and shoulder verification. These steps are described below in detail.

Multilevel Scanning: The goal of this step is to determine the centers and radii of minimum enclosing circles of all the potential heads. Typical computer vision solution for this purpose is to detect Canny edges followed by a Hough circle transformation as shown in Figure 5(b). However, the Hough circle transformation suffers significantly due to noise and cluttered background. As shown in Figure 5(b), it detects many false heads. Also, both processes are computationally expensive. So, instead, we introduce a novel approach called *multilevel scanning*, and determine the centers and radii of the heads by detecting contours at different height levels (Figure 6(d)).

The average height of an adult male is about 5'7" to 5'11" and a female is about 5'2" to 5'7". As we estimate the floor depth (Section II-B), we start scanning depth data from 6' (from the floor) to 2' at 6-inch intervals. We choose these parameters in a conservative way so that we do not miss humans. When we scan depth data a height level, we discard all the depth data below that level. Figure 6(a), 6(b), and 6(c) show the depth data after scanning at levels 5', 4'6", and 4', respectively (discarded pixels are shown black). As an example, when we scan at 6', if $Person_A$ and $Person_B$ have heights 6'6" and 6'1" respectively, we only see the top 6" and 1" of their heads, respectively. We find all the contours at that depth level. For each contour, we find the minimum enclosing circle using an iterative algorithm. The center and radius of the minimum enclosing circle is considered the center and radius of the head. For each detected center and radius, we verify whether it is a person by verifying the presence of a head and a shoulder (described next). Note that a single person can be detected at different levels. In order to avoid this, we scan from the top and when we verify a person at a higher depth level, we discard all the nearby centers at lower levels.

We leverage two strategies to speed up processing. First, when we perform multilevel scanning, it is performed out of order. Instead of scanning from top (6') to bottom (2') in

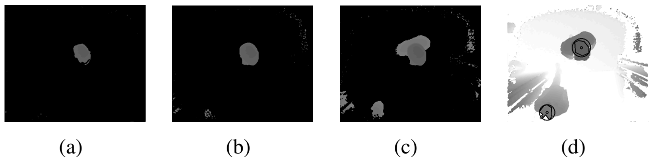
Fig. 6. Multilevel scanning at 5'(a), 4'6"(b), and 4'(c). Determining centers and radii of heads (d).

a serial order, we scan at the top most level first and then at the most bottom level, and then at the remaining levels. The intuition is that if there is someone there, FORK should capture a body at the bottom level scanning. If the bottom level scanning returns that there is nothing there compared to the approximate background (described next), we move on to process the next frame. Otherwise, we scan the remaining levels in a serial order (top to bottom) to determine the precise location of the head. Second, we do not scan at the levels that do not have enough depth data compared to the background. We do not determine the exact background as background detection often requires training and even if we take two snapshots of the same background, they tend to vary in depth data. We determine an **approximate background** by building a histogram of depth data points at different scanning levels (6-inch bin sizes). Each time we see a new frame, we update the histogram by assuming that the minimum number of depth data points seen so far at a level is from the background, which reasonably captures the wall, door, tables etc. in the environment. This approximate background detection technique enables us to move on to the next frame quickly when there is no one in the scene.

Head Verification: Given a center (cx, cy) and a radius r of a head, the goal of this step is to verify if there is a human head at this position. There are two challenges to this. First, there is a mismatch of units in (X, Y) (measured in pixels) and Z coordinates (measured in depth in millimeters) in using the depth data. To deal with this, either we transform one co-ordinate system to another, or we build a head model with two different types of units. Second, the look of a head depends on the orientation of the person. To deal with this, either we build different models for different head orientations and check them all, or we build an orientation invariant head model. In order to reduce computational complexity, we use a hemi-ellipsoid (top half of an ellipsoid) to model a human head (Figure 7), which is orientation invariant. Also, it can have two different units in the axes. An ellipsoid in Cartesian coordinates is represented by equation (1), where a, b, and c are the lengths of the semi axes and (c_x, c_y, c_z) is the center of the ellipsoid.

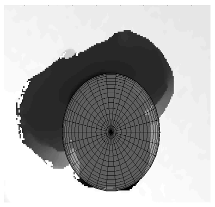
Fig. 7. Modeling a head using a hemi-ellipsoid.

$$\frac{(x-c_x)^2}{a^2}+\frac{(y-c_y)^2}{b^2}+\frac{(z-c_z)^2}{c^2}=1 \qquad (1)$$

Since we need two different units in the axes, we set a = b = r (in pixel co-ordinate), and we set $c = 0.5 * D$ (in depth co-ordinate), where D is the depth of a human head. Based on the average length of a human head [5] [4], we set D = 220 mm. We set c_x = cx, c_y = cy, and c_z = T + $0.5{*}D$, where T is smallest distance between the Kinect and the head. We iterate over the x, y values of the detected contours and use equation (1) to compute a z value for each (x,y) and compare it with the corresponding z value in the depth frame. If the average difference is less than a threshold T_{head}, we report that a head is detected. If T_{head} is too small, a small variation in a head (e.g., a pony tail) will cause us to miss it. If T_{head} is too big, something that is not a head (e.g., a box) will be reported as a head. We analyze hundreds of GBs of data and set T_{head} to 40 by performing extensive empirical studies by considering kids, people wearing turbans and caps, women with pony tails, and movement of empty chairs, boxes and a table lamp.

Shoulder Verification: Given a center (cx, cy) and a radius r of a head, the goal of this step is to verify if there is a human shoulder close to this position. It is tricky to verify, as we not only have to deal with orientation, but also with occlusion. Because, a part of the shoulder may be occluded due to the head (Figure 8(e)). In order to verify a shoulder, we go through four steps. First, we consider a region of interest (ROI) surrounding the head and the shoulder. The end-to-end distance between the two shoulders of a person is around three times his head's diameter [5] and hence we choose a slightly bigger square ROI around the head. Figure 8(a) shows one such ROI. Second, we extract the head from it by discarding all the depth data higher than T + D (computed in the head verification step), as shown in Figure 8(b). Third, we subtract the latter (Figure 8(b)) from the earlier one (Figure 8(a)) to obtain the shoulder depth data (Figure 8(c)). Note that from the first step, we discard all the depth data higher than T + D + S by setting these values to 0, where S is the depth of the shoulder. We set S to 250 mm, as $\sim$10 inch depth is reasonable enough to capture a shoulder. At step 4, we determine whether the depth data obtained from step 3 conforms to a shoulder by trying several techniques. For example, we detect contours and measure a goodness of fit to an ellipse. However, this approach suffers from occlusion (Figure 8(e), 8(f), 8(g), and 8(h)) and the surrounding environment, e.g., doors, walls, and nearby people (Figure 8(i), 8(j), 8(k), and 8(l)). Hence, at step 4, instead, we compute a histogram of depth data at different height levels and check if there is at least one bin at the shoulder depth level around the head's position that has enough depth data points to represent a shoulder. If there is no shoulder, e.g., for a ball, the depth data at that position will be close to the floor level and the bin at the shoulder level will not have enough depth data points. The purpose of the shoulder verification is to avoid spherical objects, e.g., balls, balloons, and spherical lamps. For counting people, the head verification usually suffices. However, shoulder size is a useful feature for identifying and tracking occupants.

D. Door Detection

Many existing computer vision based occupancy estimation solutions [30] [19] [21] [18] require training, where the door location or the region of interest needs to be entered manually.

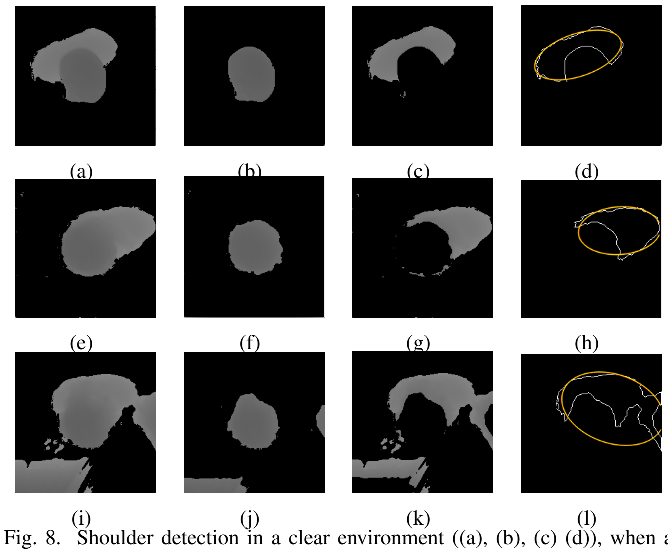

Fig. 8. Shoulder detection in a clear environment ((a), (b), (c) (d)), when a head occludes a shoulder ((e), (f), (g), (h)), and when a subject is close to a door and a nearby person ((i), (j), (k), (l)).

Our solution suggests orienting the X axis of the depth sensor parallel to the door (shown in Figure 1(a)). We use this constraint to determine the location of the door automatically, in six steps. First, starting with the preprocessed image, we do median filtering with kernel size 5 (Figure 9(a)). Second, we discard the depth data that are very close to the ground (within 1 foot) and 2 feet above it by replacing these with the maximum floor depth (Figure 9(b)). It helps us by getting an edge near the floor (marked as Special Edge, SE in Figure 9(c)) that we leverage in the subsequent steps. Third, we detect Canny edges to increase contrast and reduce noise (Figure 9(c)). Fourth, we perform Hough line transformation on the Canny edges to detect straight lines (Figure 9(d)). Even though Canny edge detection and Hough line transformations are not computationally cheap, it doesn't degrade the real-time performance as door detection is performed only at the beginning. Fifth, from the candidate Hough lines, we choose the line having the highest accumulator vote that is most parallel to the X axis of the depth frame. This line is shown as grey and all other candidate lines are shown as white in Figure 9(d). We call it $Door_1$. It is shown as line AB in Figure 9(e). Note that someone can enter into the room from the left/right side without crossing AB. To address this case, we add another door called $Door_2$, which is shown as lines CD, DE, and EF in Figure 9(e) at the sixth step. The locations of CD, DE, and EF are fixed for now, based on our empirical study on nine doors. We push CD and FE as far as possible and make sure we can see a partial head at least once at the outside. If the door is too wide to detect AB, we set AB to be the middle line of the frame. A trainer can update the door locations, if needed.

E. Tracking

FORK performs two types of tracking: (i) basic tracking to determine whether people went inside or outside through a door to count them accurately, and (ii) biometric tracking to identify and track individuals as follows.

Basic Tracking: When using RGB images, tracking is usually performed by using colors and assuming that the color of a person remains the same as he moves. When using depth images, tracking is usually performed by assuming the speed of an object changes smoothly in neighboring frames, e.g., there is no big jumps in the speed or a person stays closer to his position in the previous frame than others [28]. While these assumptions hold at a high frame rate (~30 FPS), performance degrades when the frame rate is low. There are other complex solutions that use a Kalman filter and Hungarian algorithm that are not computationally cheap. We design and implement a **lightweight greedy bipartite matching algorithm** by leveraging the position, height, and head radius of people.

Assume that we detect N and M people in the previous and current frames, respectively. For each pair of people (i,j), where $i \in \{1, 2, 3, ..., N\}$ and $j \in \{1, 2, 3, ..., M\}$, we normalize the distance between head centers, the difference of head radii and heights of each pair. Then we compute a weighted distance by using these three distances (weight: 1, 1, and 0.5, respectively). The reason for a smaller weight for height difference is that we observe that the height of a person varies up to 40 millimeters when he walks from one side to the other. Then we sort the distances in ascending order and pair them in that order. If someone $j \in \{1, 2, 3, ..., M\}$ is not paired, we add him in the current frame. However, if someone $i \in \{1, 2, 3, ..., N\}$ is not paired, we do not immediately discard him, because, it is possible that we may miss someone in a frame and detect him in the next frame. For the missing person, we **predict** the person's current position based on his average walking speed and direction, and update the location of the center of his head accordingly. To do so, every time we pair, we update average walking speed and direction of the person.

At low frame rates, someone can move a considerable distance between consecutive frames, which impacts tracking negatively, e.g., when someone (P_1) leaves through a door and someone else (P_2) enters from the other side of the door in the next frame. It may look like P_1 has moved towards his opposite direction and may increase/decrease the occupancy count erroneously. As the head of P_1 is missing at the current frame, the greedy bipartite matching tries to match the earlier frame's P_1 with the current frame's P_2. To avoid this, we consider the walking direction and if the matching requires a reversal of direction, we check if there is a presence of a ***depth shadow*** of P_1 in the current and previous frames at his respective predicted positions. By *depth shadow*, we mean a head is missing, but a partial body contour is seen near to that location. If a *depth shadow* is present, we assume that P_1 is/was there while P_2 enters and we do not allow the matching.

Biometric Tracking: Every time when someone enters/exits, FORK extracts 38 simple features regarding height, head radius, shoulder size, going in/coming out, and walking speed of the subject. More specifically, for height, FORK extracts 12 features including the minimum, maximum, average, and exact height from the depth data when s/he is crossing $Door_1$,

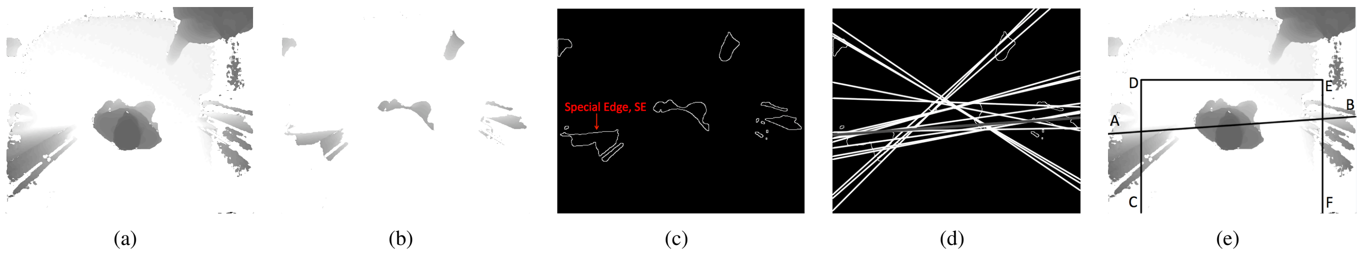

Fig. 9. Door detection. After preprocessing and median filtering (a), depth data between 1 and 2 feet from the ground (b), Canny edge detection (c), Hough line transformation (d), and door detection (e).

$Door_2$, and overall minimum, maximum, average, and median height during the entrance/exit event. Similar features are extracted regarding the head radius and shoulder size. Several machine learning algorithms are trained using these features to identify individuals (c.f. Section IV-D).

F. Counting

For each frame, for each person within that frame, we determine D_i, which is 1 if he is at the left side of $Door_i$ and 0 otherwise, where $i \in \{1, 2\}$. To be at the left side, someone's head center has to be to the left of the line segment AB for $Door_1$, and to the left of all the three line segments CD, DE, and EF for $Door_2$. We increase the occupancy count if someone's D_i is changed from 1 (at the previous frame) to 0 (at the current frame). We note the direction of the person and if his D_j ($j \neq i$) is changed from 1 to 0 later, we do not increase the count again. However, if either D_i or D_j is changed from 0 to 1 later, we decrease the occupancy count and ignore a similar change (0 to 1) subsequently.

III. Deployment & Data Collection

We deployed five instances of FORK at a Bosch office and four instances of FORK at Scaife Hall on the CMU campus. At the Bosch office, we deployed the units to cover the main office area (requires monitoring at two entrances: main entrance [Figure 1(a)] and a remote entrance), two conference rooms (Warhol and Clemente), and a lab. At Scaife Hall, we deployed the units in two classrooms (room 212 and 220) and an auditorium (room 125), which had a double door and required two FORK units. The size of the doors varied from 3 feet to 6 feet. The office had a 9.33 feet drop ceiling. CMU deployments had similar heights (9.43 feet to 9.48 feet). The units were deployed on August 24th 2015 (four instances at Bosch), December 15th 2015 (four instances at CMU), and January 27th 2016 (one instance at Bosch). All nine instances were running and we kept collecting data (people count) till June 7th 2016, when we took down the Bosch units for office relocation. The four instances at the CMU are still running. We have collected over nine months of occupancy data from these deployments. We plan to analyze this long term occupancy data to determine how we can use it for energy efficient HVAC control in commercial spaces and academic buildings in the future. We also collected over 750 GB of depth data from Scaife auditorium 125 for over a week and obtained over 3.65 million depth frames containing thousands of human heads, which was used to determine FORK model parameters. We also collected over 250000 depth frames from the Bosch office for this purpose.

IV. Evaluation

We evaluate the performance of FORK in terms of its ability of human detection, occupancy estimation at realistic scenarios, occupant identification, and door detection. We also compare its performance with break-beam sensors.

A. Experimental Setup

We evaluate FORK in all the nine deployment settings in order to determine its performance in door detection. In order to evaluate its performance in occupancy estimation, we need the ground truth of occupancy information. We consider traffic and door size, and choose three doors for this purpose. We choose Bosch office main entrance (Figure 1(a)) and CMU Scaife Hall 220 for high traffic. Both are three feet wide doors. We choose Bosch lab door since it is a six feet wide door and use it to evaluate how performance degrades at wider doors (c.f. Section IV-I) and how FORK performs when multiple people enter and exit simultaneously (c.f. Section IV-C1). Note that ground truth is obtained by a human observer standing around 10 feet distance from the door, who compares FORK's reported count with actual count in real-time without using a video camera. We keep two human observers to make sure at least one is present during the entire time of evaluation even if there is no traffic and the room being monitored is empty.

B. Human Detection Performance

In this section, we evaluate the performance of FORK in terms of its ability to detect heads, shoulders, and humans. Even though we collected over 3 million depth frames (c.f. Section III), since human verification requires ground truth of the location of heads and shoulders, we use 4100 frames for this evaluation. These frames contain a total of 2505 humans. The ground truth of the centers and radii of their heads is labeled by a human labeler. Assume that FORK verifies a head at center (x_1, y_1). If there is a center of a human head at a position (x_2, y_2) with radius r in the ground truth at that frame and if $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2)} < \frac{r}{2}$, we consider it a true positive (TP). If there are no human heads in the ground truth with that proximity, we consider it a false positive (FP). If FORK verifies that it is not a head and there are no heads nearby in the ground truth, we consider it a true negative (TN). If there is a head within that proximity in the ground truth, we consider it a false negative (FN). Then we compute precision =

Detection	Precision(%)	Recall(%)	F-score(%)
Head	98.10	96.51	97.30
Shoulder	88.77	99.21	93.70
Human	98.12	95.80	96.95

TABLE I
HEAD AND SHOULDER DETECTION PERFORMANCE

$\frac{TP}{TP+FP}$, recall = $\frac{TP}{TP+FN}$, and F-score =$2 \cdot \frac{precision \cdot recall}{precision+recall}$ of head detection. Similarly, for shoulders, when FORK verifies a shoulder, we compare with the ground truth to see if there is a shoulder there and compute precision, recall, and F-score accordingly. If both the head and shoulder verification pass, FORK reports it as a human.

We show the precision, recall, and F-score of detecting heads, shoulders, and humans (when both head and shoulder verification pass) in Table I. The results show high precision, recall, and F-score of human detection. The head detection precision and recall are very high. Sometimes the head detection fails when the heads lie near the edges of the frames. The precision of shoulder detection is a bit low since the shoulder verification passes sometimes when there is a door or a wall nearby. However, in these cases, head verification reports that there is no head, which enables us to have high precision for human detection.

C. Real-Time Occupancy Estimation Performance

1) Comparison with state of the art: We evaluate the performance of FORK in terms of its ability to estimate occupancy in real-time and compare with that of state of the art break beam sensors (All-Tag Bi-directional people counter with display [1], costs \$285 per unit). We do the evaluation at Scaife 220 from 9:00 AM to 5:15 PM. It is a real test for FORK as no depth data containing the occupants of this classroom has been collected before. The way ground truth is collected is described in Section IV-A. There are 146 entrance events and 146 exit events. There are many realistic scenarios that both solutions need to address, e.g., door opening and closing, students coming with backpacks, gym bags, papers, laptops, bike helmets hanging around the shoulders, carrying jackets in hands, carrying paper scrapbooks, wearing headphones over their heads, wearing sunglasses over their heads, wearing caps, and wearing hoodies. The performance results are shown in Table II. FORK processes data at 9 FPS on average (without median filtering) and detects all 146 entrance events accurately, achieving 100% accuracy. Among 146 exit events, it detects all the exit events accurately, except it reports twice for one exit event, achieving 99.32% accuracy. We can not explain why FORK makes that error as we have not saved the images and have not changed FORK's code to keep a log of its analysis, which would hurt FORK's processing rate and its performance. If it were a post-facto analysis, we could have identified the reason of error more easily. But then the result would not show FORK's real-time performance. Break-beam sensor makes 2 errors in detecting entrance events and makes 6 errors in detecting exit events, achieving 98.63% and 95.89% accuracy, respectively. The causes of these errors are students peeking into classes quickly without going in, a few students having conversation at the doorway, students playing with the break-beam sensor out of curiosity, multiple people leaving simultaneously counted as one, and a single person leaving counted twice. The reason for more errors at the exit events is that students' arrivals are sparse when a class starts, but a large portion of the students leave very shortly after a class is over. Note that we mount the break-beam sensors 4.2 feet above from the ground in order to detect adults according to the manual. Deploying it at a higher level will cause it to miss shorter people and deploying at a lower level may cause it to count just legs. We notice that deploying it at such a level may not be robust, as people carrying backpacks can hit the sensors inadvertently, especially in a narrow door. For a wider door, break-beam sensors can not detect multiple people entering/leaving simultaneously, as shown below.

Events	Ground Truth (# of events)	FORK (# of events)	FORK Accuracy(%)	BB (# of events)	BB Accuracy (%)
Entrance	146	146	100	144	98.63
Exit	146	145	99.32	140	95.89

TABLE II
ACCURACY OF OCCUPANCY ESTIMATION

In order to see how FORK and break-beam sensors perform when multiple people enter/exit simultaneously, we perform a controlled experiment. We ask 2, 3, 4, and 5 people to enter, exit, and cross each other 10 times each through a 6-foot wide door (Bosch lab door). The subjects are asked to stay as close as possible. The heights of the people are 5'6", 5'7", 5'7", 5'8", and 6' (in order of their appearance in the scenario). FORK runs without median filtering at ~9 FPS. The way ground truth is collected is described in Section IV-A. The Break-beam sensor is mounted at the same height as before. The percentage of times when FORK and the break-beam sensor accurately count all the people entering and exiting are shown in Table III. It shows that FORK performs much better than the break-beam sensor if the door is wider and several people enter and exit simultaneously. FORK achieves 100% accuracy when multiple people enter/exit simultaneously. However, the performance degrades when people cross each other. This is because when people cross, they reach the edge of the door during crossing, which contains noise and that makes it hard to see heads. Also, the tracking algorithm makes mistakes when multiple people bump into each other simultaneously. The break-beam sensor only counts accurately when people enter/exit one-by-one. Multiple people aligned across the door are counted as one during entering and exiting. When people cross each other, it becomes non-deterministic. This particular break-beam sensor has a bidirectional count and in 62.5% of cases when people cross each other, neither the entering nor the exiting counter is increased. In other cases when people cross, it either counts the entering or the exiting people, but not both and even then multiple people aligned across the door are counted as one.

2) Performance at a low frame rate: We perform median filtering, which throttles FORK's processing rate to 4 FPS and evaluate its performance in occupancy estimation in real-time. We evaluate it at the Bosch main entrance (Figure 1(a)) from 12:00 PM to 6:15 PM on a work day. Note

# of People	Event	FORK Accuracy (%)	BB Accuracy (%)
2	Entrance	100	0
2	Exit	100	10
2	Crossing	100	10
3	Entrance	100	0
3	Exit	100	0
3	Crossing	90	0
4	Entrance	100	10
4	Exit	100	10
4	Crossing	80	0
5	Entrance	100	0
5	Exit	100	0
5	Crossing	90	0

TABLE III
PERFORMANCE AT MULTIPLE PEOPLE WALKING SIMULTANEOUSLY

Events	Ground Truth (# of events)	FORK (# of events)	FORK Accuracy(%)
Entrance	207	207	100
Exit	235	232	98.72

TABLE IV
OCCUPANCY ESTIMATION AT A LOW FRAME RATE

that median filtering is not needed here as the depth data is not so noisy and we see a similar performance without median filtering. The way ground truth is collected is described in Section IV-A. FORK deals with many realistic scenarios during the evaluation including door opening and closing, arrival of visitors, arrival of food caterers with boxes of food in hands, multiple people walking together, people moving with bicycles, people carrying laptops in their hands, people walking while drinking water from a bottle, people walking while talking on a phone, people waving hands over their heads, people wearing regular caps, and the arrival of a cleaning lady with a large drum of cleaning equipment. The result of occupancy estimation is shown in Table IV. FORK detects all 207 entrance events accurately and achieves 100% accuracy. Among 235 exit events, FORK detects 232 of them and achieves 98.72% accuracy. The three cases that FORK misses are when two interns try to fail the system by jumping together to exit (counted as one), when someone walks out extremely fast, and when someone covers his head by waving hands to defeat the system. The average accuracy is 99.36%.

D. Occupant Identification Performance

In order to determine if FORK can identify room occupants (*who* entered/left), we ask 11 subjects to go through a door and come back 10 times each. This provides us $11 \cdot 2 \cdot 10 = 220$ data points with ground truth that are used in this *post-facto* analysis. Every time when someone enters/exits, we extract 38 simple features as described in Section II-E. We perform a 10 fold cross validation using these features of 220 entrance/exit events and obtain 97.27% accuracy in occupant identification using the Naive Bayes classifier. The accuracy is 95%, 96.36%, and 98.18% for multilayer perceptron, random forest, and K* classifiers, respectively. In order to see how much training is needed, we vary the number of training instances from 1 to 18 and show occupant identification accuracy of the rest of the data in Figure 10 for all these four approaches. It shows that with only 6 samples, the accuracy is over 80% for all of the approaches with K* classifier reaching 91.56% accuracy. With 10 samples, all the classifiers reach over 90% accuracy.

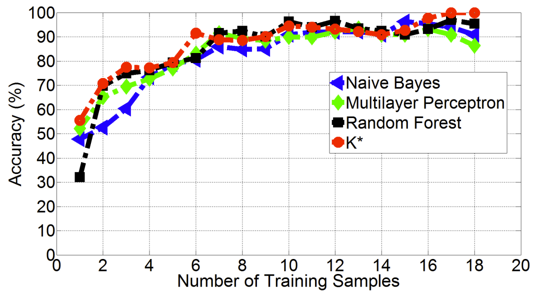

Fig. 10. Accuracy of occupant identification.

K* reaches 100% accuracy with 17 training samples. It shows a great potential for personalizing room environment, e.g., room temperature, light intensity by identifying individuals in a medium sized office. Coupled with building floorplan and multiple hypothesis testing, these features can also be used for tracking individuals throughout a building in real-time, which we leave to future work.

E. Door Detection Performance

In this section, we evaluate the performance of FORK in detecting doors. We collect data from all nine deployments and use 40000 frames for this evaluation. We consider door sizes from 3 feet to 6 feet, door opening and closing, keeping the door open and closed, doors with both (left, right) sides seen as well as one side unseen by FORK (12 feet double door in Scaife 125). We do not require keeping the space empty for door detection. In fact, people keep walking in these frames while we detect doors. As discussed in Section II-D, lines CD, DE, and EF are fixed for now. In this evaluation, we try to determine how accurate we are in detecting line AB.

The results are shown in Table V. It shows that FORK can detect doors with 100% accuracy in most instances. FORK is usually not affected by people walking through the doorway as it only considers data points between 1 and 2 feet from the ground. However, if they stand in a way that obstructs a nearby wall/door so that Canny edge detection doesn't produce special edge SE, FORK can't find the exact line AB. Also, people's movement may cause the line AB to angle a bit especially if FORK can't see the other side of the door (if the door is too wide or the edge produced by the wall is too short). Usually it is not inaccurate enough to affect occupancy estimation. However, we can overcome this limitation by detecting doors in several frames followed by a voting. Sometimes objects in the scene can affect door detection performance. In Scaife classrooms 212 and 220, there was a chair placed in a way that blocked SE and in 220 there was a half pad chair placed on the other side of the door that produced a stronger edge than the wall and caused inaccuracies. Even if someone verifies the door detection and updates if needed, the effort is minimal.

F. Impact of Different Factors on Execution Time

In this section, we demonstrate the impact of processors, OpenKinect driver, and number of people on FORK's execution time. We show the performance in terms of frame rate (number of frames processed per second) instead of execution time, as frame rate captures the real-time processing ability.

Door	#of Frames	Accuracy(%)
Lab	5000	100
Warhol	5000	100
Clemente	5000	100
Main Entrance	5000	100
Remote Entrance	5000	100
Scaife 212	2000	94.45
Scaife 220	3000	93.57
Scaife 125 (left)	5000	100
Scaife 125 (right)	5000	100

TABLE V
ACCURACY OF DOOR DETECTION

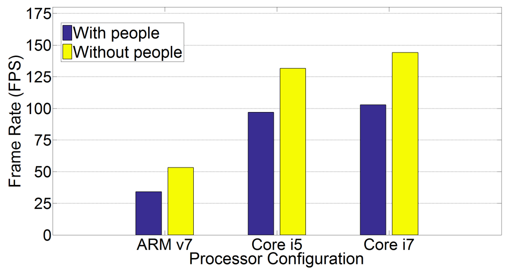

Fig. 11. Frame rate on different processors.

Impact of different processors: In order to show how fast FORK processes, we also benchmark performance on Intel® Core™ i5 and Core™ i7 processors. We use the collected data from Scaife auditorium 125 (c.f. Section III) in this analysis. We choose 2000 frames containing a dense scenario, where over 40 students came out from the auditorium in only 2.75 minutes. Figure 11 shows the processing rate (Frames Per Second) when the same solution is run on ARM v7 (in Odroid XU4), Intel® Core™ i5, and Intel® Core™ i7 processors. We run the same program 10 times in each machine and show the average result. Note that hardware configurations (eMMC card in ARM v7 vs. SSDs in the others) and operating systems (Ubuntu 15.04 on ARM v7 vs. Ubuntu 14.04 on Core™ i7 vs. Mac OSX 10.10.1 on Core™ i5) are not exactly same in all three. Still it shows the fact that FORK is extremely fast as it processes at 144.05 FPS, 131.62 FPS, and 53.2 FPS on Core™ i7, Core™ i5, and ARM v7 processors, respectively, when no one is in the scene. When there are people, it processes at 102.88 FPS, 96.93 PFS, and 34.08 FPS on average on Core™ i7, Core™ i5, and ARM v7 processors, respectively. It shows that FORK is extremely fast (102.88 FPS) compared to existing computer vision based people counting solutions that can process about 30 FPS on Core™ i7 processors.

Impact of the driver and of feeding sensor data in real-time: We use the OpenKinect driver to access the depth frames with no change in the driver code. We see a significant impact of the driver in the FORK's processing rate due to processing and filtering pixels at different stages, buffering, and I/O by OpenKinect. In order to see the impact of the driver, we feed real-time depth data using the Kinect and skip all the processing after getting a depth frame. In that case, an ARM v7 processor can process only at 13.36 FPS, which is much lower than 30 FPS in the Kinect specification. It is 98.4 FPS for an Intel® Core™ i5 processor. In another test, when we do all the subsequent processing, FORK runs ~5x slower in an ARM v7 processor when depth data is fed from the sensor in real-time compared to when (previously collected) stored depth frames are fed from an eMMC card (9.7 FPS vs 50.2 FPS). In several works [28] [10] [30] [26], frame rate is computed by analyzing stored images. It shows that frame rate computed in such a way is not the same as that of feeding depth data from the sensor in real-time.

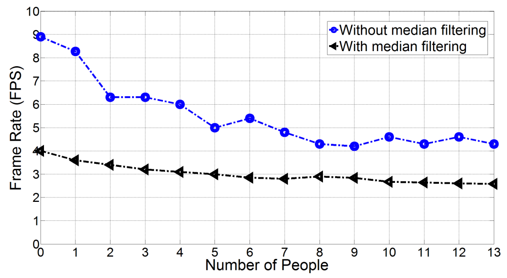

Fig. 12. Frame rate with different people count.

Impact of number of people: When there are more people in the field of view, FORK's frame rate degrades as they require more head and shoulder verification. Figure 12 shows how frame rate varies when the number of people are changed from 0 to 13. When median filtering is not used, FORK processes at 8.9 FPS when no one is in the field of view. When there are 13 people, the frame rate drops to 4.3 FPS, which is still good enough to detect and track people. However, when median filtering is used, FORK can only process at 4 FPS when no one is there. With 6 or more people in the field of view, the frame rate drops below 3 FPS, which degrades its real-time ability of occupancy estimation as shown in an experiment below.

G. Impact of Frame Rate on Occupancy Estimation

We evaluate the performance of FORK at different frame rates. We collect data at 20 FPS from the Bosch lab door deployment and use 9000 frames in this evaluation. We ask the participants to go wild as they cover their heads while walking, wave their hands, shake their heads back and forth, run fast, move table lamps and empty chairs, and stand still for a while near the door. There are 75 entrance and 75 exit events. We vary the frame rate from 20 to 0.5 FPS and show how that affects precision, recall, and F-score of occupancy estimation in Figure 13. At 4 FPS, the precision and recall are 94.23% and 98%, respectively. At 3 FPS, the precision and recall are 96.64% and 96%, respectively. It shows that we can estimate occupancy with high precision and recall with a cheaper depth sensor having a lower frame rate.

H. QoS Parameters and Real-time Analysis

FORK is a soft real-time system. Its performance depends on application requirement, e.g., occupancy estimation accuracy, selection of hardware within the cost budget. Frames are processed as fast as possible, where the achieved frame processing rate depends on several parameters, including the number of individuals being tracked. Formally, the time needed per frame consists of:

- *Data acquisition and transfer time:* It depends on image size and it is constant.

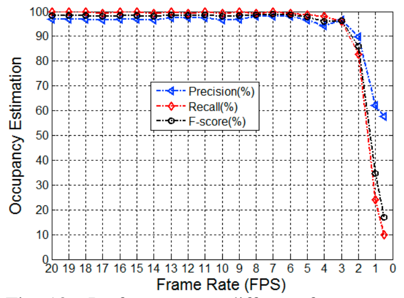

Fig. 13. Performance at different frame rates.

- *Preprocessing time:* It depends on image size and it is constant.
- *Time for multilevel scanning:* If there is no one in the scene, it is constant. Otherwise, it depends on number of people in the scene.
- *Time for verification of head/shoulder:* It depends linearly on the number of individuals detected.
- *Time for tracking individuals:* It depends linearly on all pair combinations of number of individuals of this frame and of the previous frames.

So, the time for processing a frame (T) consists of a constant part (T_0) and a part that is roughly proportional to the number of individuals (N) and can be estimated as $T \approx T_0 + p \cdot N$ seconds. The frame rate (FPS) is $1/T \approx 1/(T_0 + p \cdot N)$. Figure 12 suggests that T_0 is 1/8.9 second without median filtering and 1/4 second with median filtering. Also, it shows that p is (1/4.3 - 1/8.9)/13 = 0.00925 second per person without median filtering and (1/2.58 - 1/4)/13 = 0.0105 second per person with median filtering. p is slightly smaller without median filtering as depth values of some pixels are reported zeroes due to noise and hence ignored during the head-verification process, which saves a little computation time.

In addition to detecting individuals, FORK requires an individual to be seen on sufficient number of frames to enable reliable counting. As long as only a single person is in the field of view, a minimum of two frames showing the individual on each side of the door is sufficient. However, with more people entering or leaving simultaneously and more than one door, more frames are needed for the bipartite matching algorithm to track individuals reliably. Also, it depends on location of the doors, location of individuals, and the number of consecutive frames FORK misses detecting them. Based on the log of a FORK unit deployed at Scaife Hall classroom 220, we see that FORK tracks an individual around 17 frames (median value) during the entire path of the individual. It processes depth data around 9 FPS. Hence, it sees a person for 1.89 seconds to enter/exit. If a FORK unit processes frames at K FPS, it will see him/her for $K\cdot$ 1.89 frames to enter/exit.

When FORK takes too long to process frames, this manifests as a loss of its performance (i.e. counting accuracy). Increasing the number of individuals lowers the frame processing rate (c.f. Figure 12) and at the same time requires processing at a higher frame rate. We can therefore expect FORK's performance to drop abruptly when the frame rate falls below a certain threshold. From Figure 13, this threshold can be estimated to be around 3 FPS. We suggest future depth sensing based people counting systems to process frames at 5 FPS or higher for having good performance consistently.

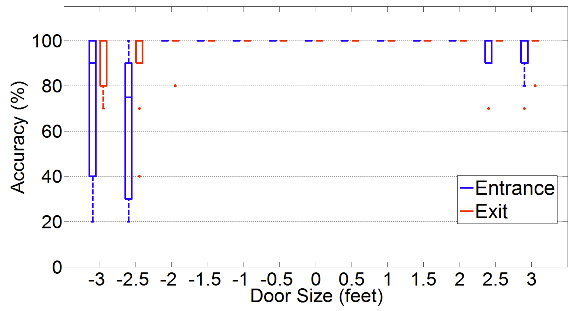

Fig. 14. Performance at different door sizes

I. Impact of Door Size

In this section, we describe how door size affects the performance of FORK. Kinect v2 has a 70.6 degrees of horizontal field of view (FOV). If it is mounted 9 feet high, it can see a 2 · 9 · tan(70.6/2) = 12.74 feet wide door at the floor level. However, since the FOV is angular and FORK requires the head to be seen, for a six feet tall person, in order to keep his head within the angular FOV, the door size becomes limited to 2 · (9-6) · tan(70.6/2) = 4.25 feet. Actually FORK can cover a bit wider door since it can detect partial heads and when people enter/exit through the edges of a door, they cross door lines CD or EF that are within this range. To determine how performance degrades for a wider door, we consider a six-foot wide door (Bosch lab door). It has a 9.3-foot drop ceiling and the Kinect is mounted at 8.9 feet from the ground. We consider 10 individuals of varying heights from 5'3" to 6'1" asking each to walk in and walk out 10 times each along straight lines that are 6 inches apart across the door. There are 13 lines and each individual generates 20 · 13 = 260 data points. Overall 208075 depth frames are collected for analysis. The box plot of the accuracy of entrance and exit events of all the individuals at different distances from the center of the Kinect co-ordinate system (marked as 0) is shown in Figure 14. This analysis reveals several interesting findings. First, we see that the accuracy is 100% at the middle and it starts to degrade at 2.5 feet from the center of the Kinect, which suggests that the performance of FORK will degrade for a door wider than 5 feet (for this mount height). There are two reasons for this: noise and missing heads. There is a significant amount of noise at the edges and sometimes even though the head is in the frame, it can't be seen due to noise (there is a big hole in the head). The noise can be asymmetric as we see more noise at the left side. Second, we see that FORK can detect shorter (height $<$ 5'5") and slender people with 100% accuracy even at the edges of the door. Short and slender people generate less noise. It suggests mounting the Kinect at a higher altitude, if possible. Taller people (height >5'9") not only produce more noise, but their heads are unseen near the edges of the door. Third, we suggest two ways to deal with noise: using median filtering at the edges of the frame (doing such in the entire frame is computationally expensive) and relaxing the error threshold T_{head} at the door edges. We use median filtering in this analysis. Relaxing T_{head} too much (> 55) causes false positives and hence is not suggested.

V. Discussions and Future Works

Currently, each FORK unit costs around \$260 (\$99 for a Kinect, \$49 for a Kinect adapter, \$74 for an Odroid, \$24 for an 8GB eMMC card, and \$10 for a WiFi dongle). A major portion of the cost is due to the high price of Kinect and its adapter. However, depth sensors have been rapidly decreasing in price as new ones become available on the market, e.g., Intel$^{®}$'s RealSense and Texas Instrument's OPT8241 TOF sensor (costs <\$60). They will be cheaper if we build one with a lower frame rate and lower resolution. FORK is more appropriate for estimating occupancy in commercial spaces, academic buildings, restaurants, and shopping centers for its cost. Hence, we evaluated FORK in one Bosch office and in one CMU campus building. Also, with this cost and high accuracy, FORK is useful for collecting "almost" ground truth for developing other solutions of occupancy estimation.

Even though FORK does not store or upload any depth image for privacy issues, biometric tracking can lead to privacy concerns. However, biometric tracking can be disabled, if needed. Note that FORK is not privacy invasive in a large space, e.g., in a shopping center, as there are many people with similar body shape there.

We could not analyze the huge amount of data we collected due to lack of resources. We plan to use the data for future projects. The FORK system has not been tested with infants and pets, which are important to detect for emergency response. We consider it future work. FORK requires almost no training for counting people as it can configure itself automatically by detecting door positions. However, for identifying and tracking individuals, it requires a little training. In the future, we will explore unsupervised machine learning algorithms to identify and track individuals. Also, in the future, we plan to use the depth sensor to determine objects that people carry while entering/exiting rooms, e.g., backpacks, laptops, boxes, and even guns, which will improve safety, security, and energy efficiency (e.g., someone leaving his office with a backpack around 5 PM may mean he is leaving for the day, whereas if he is leaving with a laptop it may mean he is going out for a meeting, and leaving empty handed may mean that he is leaving for a restroom break) using deep learning techniques. If a carefully designed model is built using a powerful machine in prior and ported to an embedded platform, the classifier can be run on the embedded platform to detect objects at real-time.

VI. Related Work

In this section, we review some of the most relevant techniques for occupancy estimation.

Break-Beam sensors: When using break-beam sensors, a pair of IR transmitters and receivers is placed across a door. The transmitters continuously transmit IR beams and if someone passes through the door, the receivers notice that the beams are broken. Based on which receiver observed the broken beam first, it can determine whether someone is entering or leaving. However, if multiple people move simultaneously (in the same direction or opposite direction), the occupancy estimation becomes inaccurate. Also, break-beam sensors can not be used for biometrics.

Ultrasonic sensors: Doorjamb [16] uses ultrasonic range finders mounted above a doorway to detect people. It can differentiate people by measuring their heights. However, it can not detect when two or more people simultaneously cross a door. [25] uses a wide-band transmitter to generate ultrasonic chirps and uses a microphone to detect changes in the reverberation to estimate the number of occupants in a room. In order to build a regression model for reverberation specific to a room, the system needs to be trained with the room empty as well as with several occupancy levels. Note that ultrasonic solutions are usually not pet friendly.

IR Array sensors: Conventional PIR motion detectors can detect human presence, but can not count the number of people. [20] places two sets of four PIR sensors to build a PIR sensor tower and use the analog signals from the PIR sensors to localize and classify a moving object. However, it can not deal with the case when multiple people move together.

Depth cameras: Depth cameras, especially the Kinect, have been used to detect, track, and count people by horizontal placement [28] [17] [10] and vertical placement [30] [26]. [26] uses a feature based approach, where it uses depth data to extract HOG features of the head and shoulder, and uses a SVM classifier for detecting pedestrians. [28] uses a model based approach to detect humans. To determine the possible positions of the heads, it uses Canny edge detectors to find all edges in the depth array and uses that to do a 2D chamfer distance matching with a binary head template. We avoid Canny edge detection in our solution because of its high computational complexity. [30] mounts Kinect vertically and finds local minimum regions in the depth image to detect humans as heads are closer to the camera than other parts of the body in this setting. However, it is not clear how it will behave if other objects (e.g., a box, a chair, or a ball) are moved under the Kinect since there is no specific checking for verifying a human head. Also, a number of these solutions [17] [10] [26] use powerful machines (e.g., Intel$^{®}$ Core^{TM} i7 processor) and/or high frame rate (around 30 FPS) for counting people whereas we tailor our solution for a low power ARM processor-based machine using depth data at a low frame rate, even at 4 FPS.

RGB cameras: A number of solutions [23] [19] [31] [29] [21] [18] [7] [12] use RGB video cameras for counting people. Several of these solutions [21] [18] [7] [12] assume that all the moving objects are people. To address that limitation, [31] counts people by detecting their faces, which is very privacy invasive compared to our solution. To detect individuals, [9] uses a flexible shape model [8] to track the silhouette of a walking pedestrian. However, [8] assumes that objects do not overlap and a reasonable proportion of the objects are not occluded. To deal with overlapping and occlusion, multiple cameras are used [11], cameras are mounted vertically looking downwards [21] [18] [7] [12]. Model based approaches [23] [19] use models to detect individuals, e.g., in [19], pedestrians are modeled as rectangular patches with an assumption that each patch is moving with a constant velocity. Feature based approaches use training based on local features, e.g., histograms of oriented gradient (HOG) [13], oriented histograms of differential optical flow [14] to detect humans. Although the reported accuracies of human detection are high for these approaches, RGB image-based approaches encounter difficulties when the background is cluttered, low illumination, or human subjects have articulated poses. In these cases, either the accuracy drops or the computation cost increases. Depth camera based approaches do not have such problems since a human body has to occupy a space regardless of the background or illumination state.

There are a few works [24] [27], where ARM processor computers are used to process RGB images. FORK is the first system that processes depth data in an ARM processor. RGB cameras/webcams are relatively inexpensive than a depth sensor. However, RGB cameras expose privacy risks if they are compromised when connected to the Internet. FORK is not as privacy invasive even if compromised as a depth sensor does not reveal the color of the clothes, skin, and hair. Also, it is hard to determine the type of clothes occupants are wearing using a depth sensor mounted in a ceiling as in FORK.

VII. Conclusions

In this work, we perform an exploratory study to understand the potential of depth sensors for detecting, estimating, and tracking building occupants. We specifically address the computational complexity issue so that the entire solution can be run on a cheaper and low power ARM processor in real-time. We evaluate and test the exploratory ideas using our prototype system FORK, which achieves over 99% accuracy in occupancy estimation at realistic scenarios. We also demonstrate the potential of identifying and tracking occupants with depth sensors using biometric features. Our extensive evaluation reveals the impact of processors, OpenKinect driver, and number of people in the field of view in the real-time performance of FORK. Our analysis demonstrates that a low frame rate depth sensor is sufficient for this application, which motivates manufacturers to build such and thus helps lower the cost of depth sensors. If depth sensors become cheaper in the future, this work serves an exploratory study to develop depth sensing based real-time systems on ARM embedded processors.

VIII. Acknowledgement

We would like to thank the anonymous reviewers and shepherd for the insightful suggestions. This work was supported, in part, by DOE grant DE-EE0007682. The opinions expressed here are those of the authors and do not necessarily reflect the views of the DOE.

References

[1] Break beam sensors from all-tag. http://all-tag.com/product-items/display-counter-bi-directional/.

[2] EIA. http://www.eia.doe.gov/.

[3] http://www.heimannsensor.com/products_imaging.php.

[4] Human engineering design data digest. http://www.acq.osd.mil/rd/hptb/hfetag/products/documents/HE_Design_Data_Digest.pdf.

[5] Human figure average measurements. http://www.fas.harvard.edu/~loebinfo/loebinfo/Proportions/humanfigure.html.

[6] Odroid XU4. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825.

[7] J. Barandiaran, B. Murguia, and F. Boto. Real-time people counting using multiple lines. In *WIAMIS*, 2008.

[8] A. Baumberg and D. Hogg. Learning flexible models from image sequences. In *ECCV*, 1993.

[9] A. Baumberg and D. Hogg. An efficient method for contour tracking using active shape models. In *IEEE Workshop on Motion of Non-Rigid and Articulated Objects*, 1994.

[10] E. Bondi, L. Seidenari, A. Bagdanov, and A. Del Bimbo. Real-time people counting from depth imagery of crowded environments. In *AVSS*, 2014.

[11] Q. Cai and J. Aggarwal. Tracking human motion using multiple cameras. In *ICPR*, 1996.

[12] T.-H. Chen and C.-W. Hso. An automatic bi-directional passing-people counting method based on color image processing. In *ICCST*, 2003.

[13] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[14] N. Dalal, B. Triggs, and C. Schmid. Human detection using oriented histograms of flow and appearance. In *ECCV*, 2006.

[15] V. L. Erickson, M. Á. Carreira-Perpiñán, and A. Cerpa. Observe: Occupancy-based system for efficient reduction of HVAC energy. In *IPSN*, 2011.

[16] T. W. Hnat, E. Griffiths, R. Dawson, and K. Whitehouse. Doorjamb: Unobtrusive room-level tracking of people in homes using doorway sensors. In *ACM SenSys*, 2012.

[17] C.-T. Hsieh, H.-C. Wang, Y.-K. Wu, L.-C. Chang, and T.-K. Kuo. A kinect-based people-flow counting system. In *ISPACS*, 2012.

[18] J. W. Kim, K. S. Choi, B. D. Choi, and S. J. Ko. Real-time vision-based people counting system for security door. In *International Technical Conference on Circuits/Systems Computers and Communications*, pages 1416–1419, 2002.

[19] O. Masoud and N. Papanikolopoulos. A novel method for tracking and counting pedestrians in real-time using a single camera. *IEEE TVT*, 50(5):1267–1278, Sep 2001.

[20] S. Narayana, R. V. Prasad, V. S. Rao, T. V. Prabhakar, S. S. Kowshik, and M. S. Iyer. PIR sensors: Characterization and novel localization technique. In *IPSN*, 2015.

[21] M. Rossi and A. Bozzoli. Tracking and counting moving people. In *ICIP*, volume 3, pages 212–216, Nov 1994.

[22] A. Rowe, M. Berges, G. Bhatia, E. Goldman, R. Rajkumar, J. Garrett, J. Moura, and L. Soibelman. Sensor andrew: Large-scale campus-wide sensing and actuation. *IBM Journal of Research and Development*, 55(1.2):6:1–6:14, Jan 2011.

[23] J. Segen and S. Pingali. A camera-based system for tracking people in real time. In *ICPR*, 1996.

[24] S. Shah. Real-time image processing on low cost embedded computers. Technical Report UCB/EECS-2014-117, EECS, UC Berkeley, 2014.

[25] O. Shih and A. Rowe. Occupancy estimation using ultrasonic chirps. In *ICCPS*, 2015.

[26] Q. Tian, B. Zhou, W. hua Zhao, Y. Wei, and W. wei Fei. Human detection using HOG features of head and shoulder based on depth map. *JSW*, 8(9):2223–2230, 2013.

[27] W. Wolf, B. Ozer, and T. Lv. Smart cameras as embedded systems. *Computer*, 35(9):48–53, 2002.

[28] L. Xia, C.-C. Chen, and J. Aggarwal. Human detection using depth information by kinect. In *CVPRW*, 2011.

[29] D. Yang, H. Gonzalez-Banos, and L. Guibas. Counting people in crowds with a real-time network of simple image sensors. In *ICCV*, 2003.

[30] X. Zhang, J. Yan, S. Feng, Z. Lei, D. Yi, and S. Li. Water filling: Unsupervised people counting via vertical kinect sensor. In *AVSS*, 2012.

[31] X. Zhao, E. Delleandrea, and L. Chen. A people counting system based on face detection and tracking in a video. In *AVSS*, 2009.