A powerful and cost-efficient human perception system for camera networks and mobile robotics

Marco Carraro, Matteo Munaro, and Emanuele Menegatti

University of Padova, Department of Information Engineering, Via Gradenigo 6/A, 35131 - Padova, Italy
{marco.carraro, matteo.munaro, emg}@dei.unipd.it,
WWW home page: http://robotics.dei.unipd.it

Abstract. In this work, we present a software library which permits the efficient usage of the Kinect One, a time-of-flight RGB-D sensor, with the nVidia Jetson TK1, an ARM-based embedded system, for the purpose of people detection. Our software exploits nVidia CUDA to obtain all the data necessary for robust people detection and other perception algorithms by parallelizing the generation of the 3D point cloud and many pixel-wise operations on both the raw depth and the infrared images coming from the sensor. The library developed has been released as open-source and the whole system has been tested as a people detection node in an open source multi-node RGB-D tracking framework (OpenPTrack). The results gathered show that the proposed system can be effectively used as a people detection node, outperforming the state-of-the-art in terms of people detection framerate not only with the nVidia Jetson, but also with non-embedded computers.

Keywords: Kinect One, people detection and tracking, mobile robotics, nVidia Jetson, OpenPTrack

1 Introduction

Human detection and tracking is a fundamental skill in many fields, e.g. surveillance, mobile robotics, ambient assisted living (AAL), culture and arts. In recent years, the advent of low cost RGB-D sensors such as the Microsoft Kinect, boosted up the research on mobile robotics, making it possible to reach new levels of accuracy in robot perception. Recently, Microsoft released the second generation of the Kinect sensor, the *Kinect One*, both increasing image resolution and improving depth smoothness and accuracy [1] by implementing the Time-of-flight (ToF) [2] technology. Nevertheless, the Kinect One is computationally eager because of the huge amount of data generated per frame (tens of megabytes per frame at 30 frames per second) and its Linux driver requires processing on a dedicated graphic card to perform data acquisition at the sensor's maximum framerate. As a result, this sensor is not likely to be used on mobile robotic of embedded platforms.

In this work, we propose a new software library that could allow to use efficiently the Kinect One sensor together with the nVidia Jetson TK1 embedded system as a perception node for people detection and tracking from RGB-D data. The resulting system is an ideal perception setup to be used for mobile robotics or in networks of many nodes, given the high level processing it allows to perform and the low power consumption of the Jetson embedded platform. This result was made possible with the development of a new Linux library for the Kinect One which exploits nVidia CUDA to process in parallel the operations on the raw data coming from the sensor. We validated our work by testing the developed system as a distributed node of a camera network which uses an open-source tracking library: OpenPTrack[3]. This library implements state-of-the-art algorithms to detect and track humans in heterogeneous camera networks and it is based on the de-facto standard robotics middleware, ROS (Robot Operating System)[4]. In summary, the contribution of this work is two-fold:

- We propose a new library which permits the usage of the Kinect One with CUDA-capable embedded systems and we demonstrate the validity of this work obtaining suitable frame-rates for real-world applications as people detection and tracking.
- We release this library as open-source[1] together with a ROS bridge [2] to make it work out-of-the-box with the most popular framework for the robotics community.

The remainder of the paper is organized as follows. Section 2 reviews the state-of-the-art of RGB-D sensors. Section 3 explains the features of the new library we developed. In Section 4, experiments are reported and results are shown in terms of acquisition and people tracking frequency. Finally, Section 5 draws the conclusions of this work.

2 State-of-the-art

2.1 RGB-D sensors

When dealing with mobile robots or complex surveillance scenarios, two dimensional information is not always sufficient to obtain reliable results of detection and tracking in real time. Furthermore, passive 3D solutions such as stereo cameras require additional processing for computing depth information and they are not able to estimate depth for lowly textured areas. For these reasons, the advent of active and low-cost 3D sensors, such as Microsoft Kinect, significantly improved the research on autonomous mobile robotics and computer vision. The first-generation Kinect is an RGB-D sensor that provides both color and depth data at VGA resolution. Depth is estimated by means of an active triangulation process[5] between an infrared pattern projector and an infrared camera, i.e.,

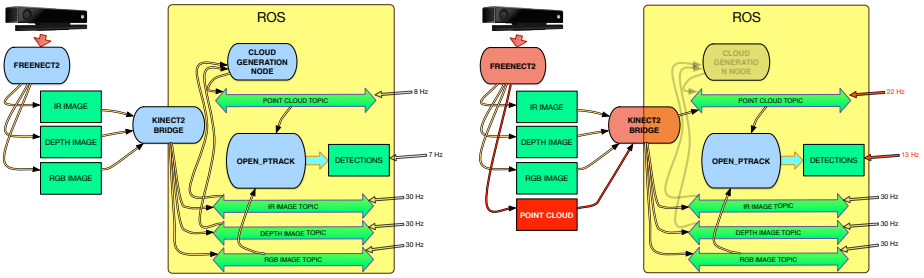
[1] `https://github.com/OpenPTrack/libfreenect2`
[2] `https://github.com/OpenPTrack/kinect2_bridge`

the position of each 3D point is the intersection of the optical rays corresponding to a dot of the projector and the one of the considered pixel in the infrared camera. The Kinect is widely used in computer vision and robotics for Simultaneous Localization and Mapping (SLAM) [6], people detection and tracking [3,7,8], short-term and long-term people re-identification [9,10,11], ambient assisted living [12,13,14] and many other applications. Besides the wide usage of this sensor, it has the drawback of not being able to estimate depth information outdoors because the infrared component of the sunlight interfers with the pattern projected by Kinect. Furthermore, the depth estimation error increases quadratically with the distance[15]. To overcome these problems, in late 2013, Microsoft released the second generation of the Kinect sensor. This new RGB-D camera relies on the continuous wave time-of-flight [2] technology to infer depth, that is an array of emitters sends out a modulated signal that travels to the measured point, gets reflected and is received by the CCD of the sensor. The sensor acquires a 512 x 424 depth map and a 1920 x 1080 color image at 15 to 30 fps depending on the lighting condition, since the sensor exploits an auto-exposure algorithm. Kinect One outperforms its predecessor on several aspects. In particular, it works outdoors up to four meters and depth accuracy remains constant while increasing the distance[1]. However, since the data resolution is higher than for Kinect v1, processing Kinect One data is computationally more demanding and turns out to be unsuitable for embedded systems. In this work, in order to overcome this problem, we modified the Linux driver and ROS wrapper for the Kinect One so that they allowed to obtain color and point cloud data at more than 20 fps with the nVidia Jetson, a CUDA-capable embedded system. While Kinect One is directly supported in Microsoft Windows with a free driver and SDK provided by Microsoft, the only driver available in Linux is unofficial and open source and is called *libfreenect2*[3]. In this work, we also used nVidia CUDA[16], a scalable library for exploiting the General Purpose GPU (GP-GPU) computing on nVidia GPUs. Our work improved a first working library[17] developed in CUDA. The numerical comparisons between the different versions of these libraries are presented in Section 4.

2.2 People detection and tracking in camera networks

The ability to autonomously detect and track humans in camera networks is one of the most important issues in robotics and computer vision applications. The problem can be split into two different sub-problems: (1) perform people detection and tracking within a view of a single camera and (2) maintain the same ID for the same person seen by different cameras[18]. For solving (1), a wide set of works in literature relies on RGB data alone[19,20], while, recently, new methods were developed for using RGB-D data to perform this task[3,21,22]. The problem of associating the correct ID among different cameras (2) is often solved based on the knowledge of camera poses and by exploiting features extracted from the person motion and appearance. OpenPTrack is an open source software

[3] `https://github.com/OpenKinect/libfreenect2`



(a) The state-of-the-art data flow

(b) The proposed data flow

Fig. 1: High level representations of the two data flows. On the left, the state-of-the-art data flow of the OpenPTrack system using the Kinect One and the Jetson embedded system. The 3D point cloud is computed outside the *libfreenect2* library by the ROS nodelet `cloud_generation_node`. On the right, the proposed data flow of the same system. The 3D point cloud is now directly computed within the new version of *libfreenect2* and streamed by the new version of *kinect2_bridge*, thus the external nodelet is no longer required.

for multi-camera calibration and people tracking in RGB-D camera networks. It allows scalable, robust and real-time person tracking using affordable off-the-shelf components, such as Kinect One, and an open source codebase. It constitutes a powerful tool for enabling interactive experiences for education, arts and culture, but it is also exploited for guaranteeing people safety in industrial environments[23]. The OpenPTrack nodes which use the Kinect One are usually equipped with a powerful computer with a dedicated GPU because the sensor is eager of performance. The use of these computers causes space problems, high costs and high power consumption. Moreover, the OpenPTrack network can potentially be made of dozens of nodes, thus amplifying these problems. Therefore, the use of embedded systems as the nVidia Jetson can fix these issues, allowing the building of large networks.

3 Methodology

Our objective is to acquire data from the Kinect One sensor with the nVidia Jetson TK1 at high frame rate and integrate the camera into an OpenPTrack network to perform people detection and tracking. The state-of-the-art Linux driver for the Kinect One, *libfreenect2*, is not able to perform the operation needed by OpenPTrack at a framerate suitable for people tracking. For this reason, in order to improve the performance, we use nVidia CUDA, shifting computational burden from the ARM CPU to the GPU of the embedded system. In Fig. 1a, the overall state-of-the-art system needed to perform people detection with Kinect One is shown. At first, the Linux driver for Kinect One, *libfreenect2*, is used to obtain the raw data from the sensor. OpenPTrack is based

on the ROS middleware, thus, in order to interface the Kinect camera with this people tracking library, we need a ROS wrapper of the *libfreenect2* driver. This wrapper is implemented in the *kinect2_bridge* ROS package, that reads sensory data obtained from the driver and streams them to ROS topics whenever a ROS node requests them. In this work, we adapt also this wrapper to be compatible with the proposed version of *libfreenect2*. The algorithm exploited by the standard version of the driver for computing depth image, infrared (IR) image and point cloud is reported in Algorithm 1, while the one we propose in this paper is reported in Algorithm 2 and illustrated in Fig. 2. The RGB information that comes from the sensor does not need any additional computation, so we store it directly without passing it to the GPU. The operations needed to transform the raw depth and infrared data in the final data needed by the tracking library are all pixel-wise or consist of operations on the neighborhood of each pixel. These types of functions are implementable in CUDA, thus lowering the final computational complexity from $O(N)$ to $O(1)$, with N number of pixels. Furthermore, in our approach, the generation of the point cloud from IR and depth data is not computed any more by the ROS wrapper, but directly within the driver, thus shifting computational burden from the ARM CPU to the GPU of the embedded system. Each parallel function we implemented requires as input the number of CUDA threads that will concurrently operate. Since the dimension of the depth and infrared images is 512x424, we designed a grid of 512 threads per block with $\left\lfloor \frac{512*424+(512-1)}{512} \right\rfloor = 424$ blocks [24]. OpenPTrack, the library we use for performing people detection, requires as input from the Kinect One a point cloud filled with 3D points *colored* with the corresponding intensity obtained from the infrared image. Here, the infrared is preferred to the RGB because the former is constant also in the dark. However, to help the people detection module, an intensity rescaling operation has to be performed on the intensity image in order to improve its contrast, thus helping people detection. Also this computation is performed in our version of the *libfreenect2* driver by exploiting the parallelization achievable with CUDA. In the next sections, we detail the important steps developed in this work.

3.1 Memory management

When exploiting GPU processing, the typical bottleneck is the overhead due to the data transfer between the central memory and the GPU memory[25]. To prevent these passages to affect the overall performance of our algorithm, we pre-allocate in the GPU memory the exact space needed by the infrared and depth images and the point cloud. This way, we avoid new allocations whenever a new frame is acquired by the sensor, thus making the GPU only overwrite the previous frame information. The memory passages performed by our algorithm consist of the copy of the input data from CPU to GPU (only IR and depth raw images) and of those needed to transfer the output data from GPU to CPU central memory at the end of GPU processing. Fig. 2 highlights the memory transfers performed by our algorithm. In particular, the memory transfers are

Algorithm 1: Standard algorithm of the `libfreenect2` driver for each frame

input : a frame $F = (I, D, M)$ where I is the raw infrared image, D is the raw depth image and M is the camera calibration matrix
output: The final infrared image $\widehat{I}$, the final depth image $\widehat{D}$ and the point cloud P

```
foreach pixel p of D do
    computeDepth(p,p̂)
foreach pixel p of I do
    computeIR(p, p̂)
foreach pixel p_d of D̂ and the correspondent pixel p_i of Î do
    computePoint(p_d, p_i, p_p)
```

less than 500 KB for the first transfer (from CPU to GPU) and the same quantity plus 7 MB (the point cloud) for the transfer-back. The maximum data transfer required by the application is then $7.5 * 30 = 225$MB/s which is about the 1.51% of the total Jetson GPU bandwidth and the 0.067% of a nVidia Geforce GTX Titan Black total GPU bandwidth.

3.2 Point Cloud generation

A 3D point cloud is the typical input of several algorithms in 3D computer and robot vision [26,27]. This data structure is needed also by OpenPTrack to perform people detection while being robust to light changes. This data type is built from three pieces of information: the depth map, the infrared image and the intrinsic parameters of the sensor. The point cloud is computed at each new frame after that the depth data have been processed and become available. Given the equations of 3D perspective projection:

$$\begin{bmatrix} x \\ y \\ d \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{1}$$

where (x, y) are the coordinates of a pixel in the depth image, d is the measured depth, f_x, f_y, c_x, c_y are the intrinsic parameters of the sensor and represent the focal lengths and the optical centers of the camera, we can obtain (X, Y, Z), which are the 3D coordinates of the correspondent point in the point cloud, with

$$\begin{cases} X = \frac{(x - c_x)d}{f_x} \\ Y = \frac{(y - c_y)d}{f_y} \\ Z = d \end{cases} \tag{2}$$

For what concerns color information, in order to maintain the same structure as for point cloud colored with RGB information, we consider all the three R,

Algorithm 2: Our version of the `libfreenect2` driver for each frame

input : a frame $F = (I, D, M)$ where I is the raw infrared image, D is the raw depth image and M is the camera calibration matrix

output: The final infrared image $\widehat{I}$, the final depth image $\widehat{D}$ and the point cloud P

1 Image_size ← 512 * 424;
2 Block_size ← 512;
3 Grid_size ← $\left\lfloor \frac{Image\_size+(Block\_size-1)}{Block\_size} \right\rfloor$;
4 memoryCopyFromCPUToGPU(D,I);
5 computeDepth⟨⟨⟨Grid_size, Block_size⟩⟩⟩(D, $\widehat{D}$);
; `// pixel-wise depth computation`
6 computeIR⟨⟨⟨Grid_size, Block_size⟩⟩⟩(I, $\widehat{I}$);
; `// pixel-wise IR computation`
7 computePointCloud⟨⟨⟨Grid_size, Block_size⟩⟩⟩($\widehat{D}$,$\widehat{I}$,M,P);
; `/* pixel-wise point cloud generation (every pixel corresponds to a point) */`
8 memoryCopyfromGPUToCPU($\widehat{D}$,$\widehat{I}$, P);

G and B channels and fill them with i, that is the infrared intensity of the pixel (x, y), in the IR image. Indeed, people detection performed on infrared information is more robust to changes in visible light. We fill the three R, G and B channels with the same i values in order to have the same algorithms working both on real RGB-colored point clouds and intensity-colored clouds. It is worth noting that, in this work, the additional computational burden due to the filling and use of three identical intensity channels is actually negligible. The space in memory allocated for the cloud is then filled in with $P = (X, Y, Z, R, G, B)$ points. Once the point cloud is filled, it is transferred back to the CPU (bottom part of Fig. 2). An illustration of this process and some examples of the resulting point cloud are shown in Fig. 3.

3.3 The `kinect2_bridge` wrapper

The `kinect2_bridge` wrapper is an executable which streams data from the `libfreenect2` driver library to the ROS topics whenever these data are requested. We had to adapt this wrapper because our version of the driver directly generates the point cloud that was computed by an external ROS node with the standard versions of the driver and the wrapper (see Fig. 1). This choice allowed to save time avoiding to allocate, transfer and fill each point cloud.

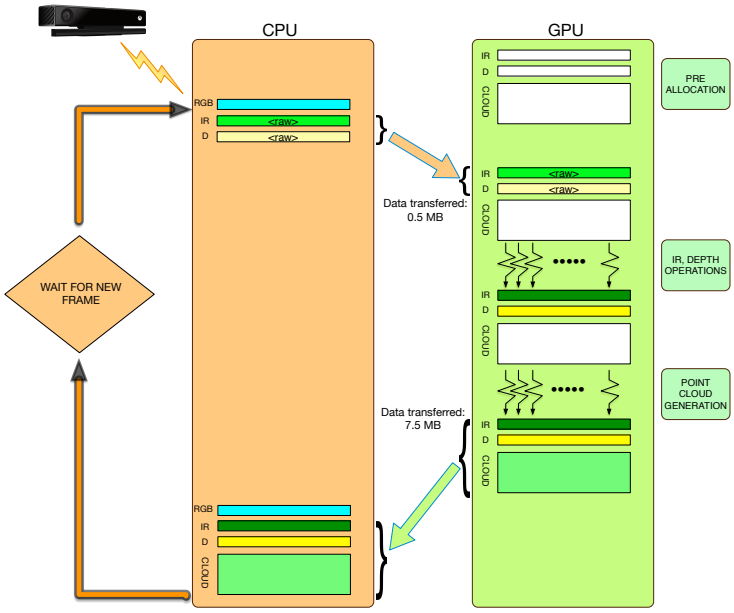


Fig. 2: The processing flow performed by our library for each single frame. At start-up, the space for the needed data is allocated once in the GPU memory, then, for each frame, the data coming from the CPU memory are processed to achieve the final data. The data transferred from CPU to GPU consist of the raw IR and depth data, while the data transferred from GPU to CPU consists of the final IR, depth and point cloud obtained after the parallel computations have been made.

4 Experiments and Results

In this section, we present the experiments we performed to validate the improvements of our work with respect to the state-of-the-art and prove that the complete system composed of Kinect One, Jetson TK1 and OpenPTrack can work as a powerful RGB-D perception node for people detection. In Tab 1, we quantitatively compare our work with the library in [17] in terms of frame rate of the streamed point cloud and of the complete people detection application when using the Jetson TK1 as processing unit. It can be noticed that the overall frame rate doubles by exploiting this work, thus reaching 13.7 frames per second that is enough for obtaining a continuous tracking of people. The publishing rate of the point cloud almost triples, thus providing 22 point clouds per second to the high-level ROS-based algorithms that could run on the Jetson. The same test has also been performed by substituting the Jetson embedded system with an high-end laptop, with an Intel i5-4210M CPU and a nVidia Quadro K1100M GPU. The frame rates reported in Tab 2 prove that our work can be used also with CUDA-enabled, non-embedded computers, increasing the overall perfor-

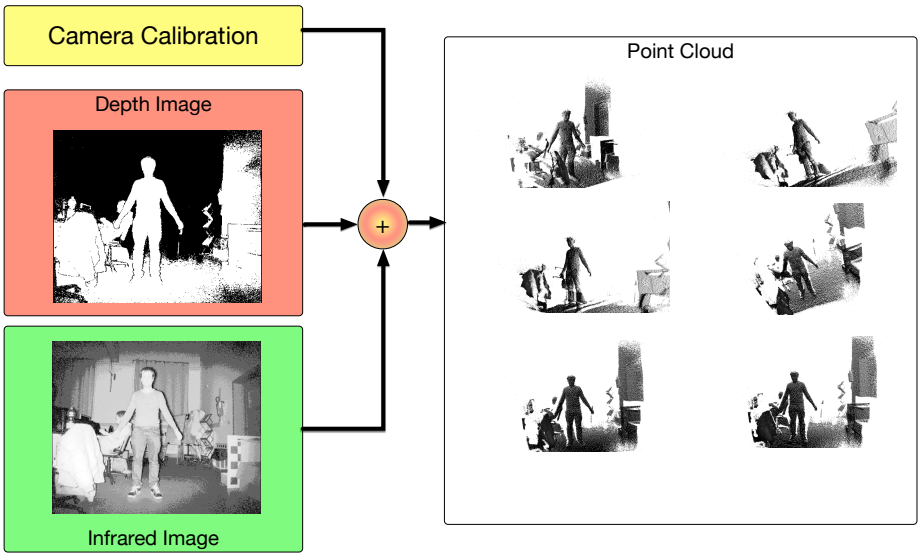


Fig. 3: The infrared point clouds are obtained from the depth and infrared images and by exploiting the intrinsic parameters of the sensor. Example of six views of the resulting cloud are reported on the right.

libraries used	Point Cloud topic framerate (Hz)	OpenPTrack detection framerate (Hz)
ours	**22**	**13.7**
[17]	8	7

Table 1: Frame rate comparison with a Jetson TK1 embedded system.

mance. Finally, we qualitatively compared the detection output obtained by the embedded system and the laptop-based system when using the libraries developed in this work together with OpenPTrack. In Fig. 4, the tracks generated in a scenario with two moving persons are reported. It can be noticed that the two persons are correctly detected by both nodes. The trail generated by the embedded node (in red) is slightly less dense than the one generated by the laptop (in green) because of the differences in frame rate outlined in Tab 1 and 2. These tests confirm that the work described in this paper allows to use Kinect One together with a CUDA-enabled embedded system as a compact, powerful and cost-efficient perception node. Furthermore, the library is useful also for mobile

libraries used	Point Cloud topic framerate (Hz)	OpenPTrack detection framerate (Hz)
ours	**30**	**25.4**
[17]	30	22

Table 2: Frame rate comparison with a high-end laptop.

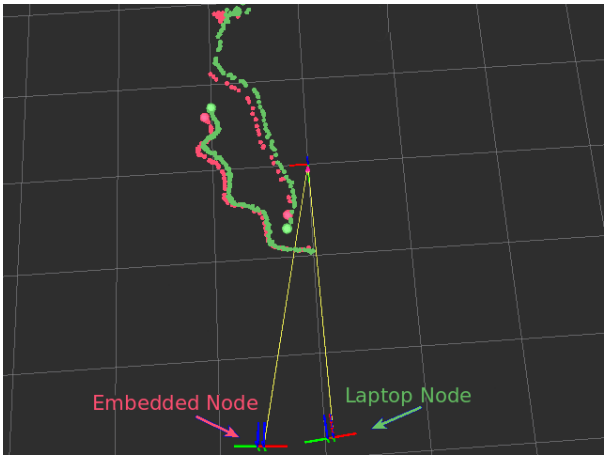


Fig. 4: Output of OpenPTrack obtained while two persons were freely moving in the scene. Each person has two trails, the green one comes from a Kinect One attached to a high-end laptop, while the pink one is generated from a Kinect One coupled with a nVidia Jetson TK1. It can be noticed that the two persons are correctly detected by both nodes. The trail generated by the embedded node is slightly less dense than the one generated by the laptop because of the differences in frame rate outlined in Tab 1 and 2. The two tracks present a small offset for visualization purposes.

robotics where real-time performance, weight and autonomy of the equipment are an issue. Indeed, the library outputs all the information needed to build robust navigation algorithms and it is compatible with ROS, the de-facto standard used in the robotics community.

5 Conclusion

In this work, we presented a powerful and cost-efficient human perception system useful for people detection purposes in camera networks and mobile robotics. The system is composed of an embedded system, the nVidia Jetson TK1, the Kinect One and OpenPTrack, an open-source, ROS-compatible, multi-camera people tracking system. The main contribution of this work is the development of a new library which exploits the GPU capabilities of the Jetson to process at high frame-rate the rich stream of data coming from the Kinect One. Another contribution is the development of a wrapper to make the library compatible with the ROS middleware. To demonstrate the validity of the perception system, we tested it while performing people detection in a network using OpenPTrack. The results are that point cloud generation is performed at 22 Hz and people detection is obtained at 14 Hz, thus tripling and doubling, respectively, the frame rates that were obtained with the state-of-the-art library [17]. This work is not

only important for systems of many nodes, where each node has to be as powerful and cheaper as possible, but also as perception system for mobile robotics, given its compatibility with ROS. To provide the best benefit for the computer vision and robotics community, but also for developers of human-computer interaction applications, we released all this work as open source within the OpenPTrack repository[4].

Acknowledgments

Portions of this work have been supported by OpenPerception and the REMAP center at UCLA. The authors would like to thank Randy Illum and Jeff Burke at UCLA for the extensive testing of the developed software.

References

1. S. Zennaro, M. Munaro, S. Milani, P. Zanuttigh, A. Bernardi, S. Ghidoni, and E. Menegatti, "Performance evaluation of the 1st and 2nd generation kinect for multimedia applications," in *Multimedia and Expo (ICME), 2015 IEEE International Conference on*, pp. 1–6, IEEE, 2015.
2. S. B. Gokturk, H. Yalcin, and C. Bamji, "A time-of-flight depth sensor-system description, issues and solutions," in *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*, pp. 35–35, IEEE, 2004.
3. M. Munaro, F. Basso, and E. Menegatti, "Openptrack: Open source multi-camera calibration and people tracking for rgb-d camera networks," *Robotics and Autonomous Systems*, 2015.
4. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, 2009.
5. K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.
6. F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the rgb-d slam system," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1691–1696, IEEE, 2012.
7. M. Munaro and E. Menegatti, "Fast rgb-d people tracking for service robots," *Autonomous Robots*, vol. 37, no. 3, pp. 227–242, 2014.
8. M. Munaro, F. Basso, S. Michieletto, E. Pagello, and E. Menegatti, "A software architecture for rgb-d people tracking based on ros framework for a mobile robot," in *Frontiers of Intelligent Autonomous Systems*, pp. 53–68, Springer, 2013.
9. F. Fleuret, H. B. Shitrit, and P. Fua, "Re-identification for improved people tracking," in *Person Re-Identification*, pp. 309–330, Springer, 2014.
10. M. Munaro, S. Ghidoni, D. T. Dizmen, and E. Menegatti, "A feature-based approach to people re-identification using skeleton keypoints," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 5644–5651, IEEE, 2014.

[4] `https://github.com/OpenPTrack/libfreenect2/tree/jetson-dev`
`https://github.com/OpenPTrack/kinect2_bridge/tree/jetson-dev`

11. L. Nanni, M. Munaro, S. Ghidoni, E. Menegatti, and S. Brahnam, "Ensemble of different approaches for a reliable person re-identification system," *Applied Computing and Informatics*, 2015.
12. M. Carraro, M. Antonello, L. Tonin, and E. Menegatti, "An open source robotic platform for ambient assisted living," *Artificial Intelligence and Robotics (AIRO)*, 2015.
13. D. Fischinger, P. Einramhof, K. Papoutsakis, W. Wohlkinger, P. Mayer, P. Panek, S. Hofmann, T. Koertner, A. Weiss, A. Argyros, *et al.*, "Hobbit, a care robot supporting independent living at home: First prototype and lessons learned," *Robotics and Autonomous Systems*, 2014.
14. S. Ghidoni, S. M. Anzalone, M. Munaro, S. Michieletto, and E. Menegatti, "A distributed perception infrastructure for robot assisted living," *Robotics and Autonomous Systems*, vol. 62, no. 9, pp. 1316–1328, 2014.
15. F. Basso, A. Pretto, and E. Menegatti, "Unsupervised intrinsic and extrinsic calibration of a camera-depth sensor couple," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 6244–6249, IEEE, 2014.
16. J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.
17. L. Xiang, "libfreenect2 CUDA library." `https://github.com/xlz/libfreenect2`, 2015. [Online; accessed 2016-02-03].
18. X. Wang, "Intelligent multi-camera video surveillance: A review," *Pattern recognition letters*, vol. 34, no. 1, pp. 3–19, 2013.
19. R. Vezzani, D. Baltieri, and R. Cucchiara, "Pathnodes integration of standalone particle filters for people tracking on distributed surveillance systems," in *Image Analysis and Processing–ICIAP 2009*, pp. 404–413, Springer, 2009.
20. A. Kandhalu, A. Rowe, R. Rajkumar, C. Huang, and C.-C. Yeh, "Real-time video surveillance over ieee 802.11 mesh networks," in *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, pp. 205–214, IEEE, 2009.
21. O. H. Jafari, D. Mitzel, and B. Leibe, "Real-time rgb-d based people detection and tracking for mobile robots and head-worn cameras," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 5636–5643, IEEE, 2014.
22. M. Munaro, F. Basso, and E. Menegatti, "Tracking people within groups with rgb-d data," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 2101–2107, IEEE, 2012.
23. M. Munaro, C. Lewis, D. Chambers, P. Hvass, and E. Menegatti, "Rgb-d human detection and tracking for industrial environments," in *Intelligent Autonomous Systems 13*, pp. 1655–1668, Springer, 2016.
24. J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.
25. S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-m. W. Hwu, "Optimization principles and application performance evaluation of a multithreaded gpu using cuda," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pp. 73–82, ACM, 2008.
26. A. Aldoma, F. Tombari, J. Prankl, A. Richtsfeld, L. Di Stefano, and M. Vincze, "Multimodal cue integration through hypotheses verification for rgb-d object recognition and 6dof pose estimation," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 2104–2111, IEEE, 2013.
27. P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.