AC 2012-4751: LOW-COST, HIGH-CAPABILITY, EMBEDDED SYSTEMS FOR EDUCATION AND RESEARCH: A TOOLBOX FOR THE MICROSOFT KINECT

joshua fabian, Villanova University
Mr. Tyler A. Young, Villanova University

Tyler Young implemented the initial interface between Simulink (in a Unix environment) and the Microsoft Kinect device.

Dr. James Peyton Jones, Villanova University

James Peyton Jones is a professor of electrical and computer engineering, and a member of the Center for Nonlinear Dynamics and Control at Villanova University.

Dr. Garrett Miles Clayton, Villanova University

Garrett M. Clayton received his B.S.M.E. from Seattle University and his M.S.M.E. and Ph.D. in mechanical engineering from the University of Washington, Seattle. He is an Assistant Professor in mechanical engineering at Villanova University. His research interests focus on mechatronics, specifically modeling and control of scanning probe microscopes and unmanned vehicles.

Low-cost, High-Capability, Embedded Systems for Education and Research: A Toolbox for the Microsoft Kinect

J. Fabian, T. Young, J.C. Peyton Jones, G.M. Clayton
Center for Nonlinear Dynamics & Control, Villanova University

Abstract

Increasingly powerful, yet low-cost computing and sensing devices are now available for use in student designs and embedded mechatronic systems. The Microsoft Kinect, for example, though initially developed as a gaming device, provides rich sensing possibilities, with camera and depth images, again at remarkably low cost. However, the sophistication of such devices often requires a high degree of programming ability in order to exploit their evident capabilities. This paper describes progress on a National Science Foundation and MathWorks sponsored project aimed at making these devices more accessible to student users through the use of Automatic Code Generation techniques. Specifically, the paper describes a new toolbox that has been developed which allows students to perform their designs from within the Matlab / Simulink environment, and then to implement these designs directly on a hardware platform coupled to the Kinect system. Students develop their designs using interconnected Simulink blocks and subsystems, and the 'build' process automatically cross-compiles and downloads the model to the target for execution. External mode capability can be used to monitor the target hardware as it executes in real time, enabling the user to tune model parameters and log data while their application is running. An example showing how the Kinect can be integrated into a higher level system design is shown as an example.

1. Introduction

Advances in low-cost, high-capability computing and sensing devices offer new opportunities for teaching, particularly in the field of mechatronics, image or signal processing and control. The use of embedded devices in the robotics area for example has been shown to increase student motivation and learning [1-3]. The LEGO MindStorms NXT, in particular, has received much interest and attention with new tools and applications in many STEM areas, [4-6]. Similar hardware advances are also resulting in the increased availability of low-cost sensing devices, including gyroscopes, accelerometers, and cameras, [7-9]. The recent introduction of the MicroSoft Kinect, takes this development one step further by providing a combined camera and depth image within a single low-cost package. Though intended primarily for the entertainment market, the Kinect has excited considerable interest, particularly within the robotics community, for its sensing potential. Early applications are largely focused on a stationary sensor detecting and/or tracking object motion in 3 dimensions, and include articulated skeleton tracking for improved human control of robots, hand gesture recognition, and 3D virtual environment construction. Recent publications include [10-12].

In all of these reports, however, it appears that the robot control and Kinect image processing algorithms have been coded in C, no doubt because the open source drivers that are available define C language interfaces. However, this presents a considerable barrier for the introduction of the Kinect in many signal processing, image processing, and control courses, since the de-facto language in many engineering colleges has moved from C to the MATLAB / Simulink environment. Furthermore, the need to resolve low-level programming details, such as fixed point arithmetic or overflow, distracts students from the higher-level pedagogical goals related to the signal processing or control algorithm. An alternative approach is for students to perform their system designs in the preferred MATLAB / Simulink environment, and then use Simulink's 'Automatic Code Generation' or 'Rapid Prototyping' capability to translate these designs into real-time executable code. The aim of this paper is to show how the Kinect device can be incorporated into high-level Simulink designs, streaming parallel camera and depth images into the user's Simulink model. The images are then readily manipulated within Simulink in order to achieve a much more sophisticated signal processing or control design than was previously possible in a classroom environment. In particular, the paper describes a new 'VU-Kinect' block which makes Kinect depth and camera images easily accessible to users in the Simulink environment. It should be noted, however, that since the start of this project, other similar Simulink-based solutions have also become available, [13-15]. The specific 'VU-Kinect' block described in the paper should therefore be regarded as one instantiation of several recent developments which enable the excitement and interactivity of the Kinect device to be brought into the classroom.

The paper is organized as follows. An overview of the hardware and software design trade space is presented in section 2, including the specific architecture used for the present research. The VU-Kinect block, which provides the interface from the Kinect device to Simulink, is discussed in detail in section 3. An object detection example is presented in section 4 and used to demonstrate the utility of the Kinect device and the VU-Kinect block. Finally, brief conclusions and plans for future work are discussed in section 5.

2. Kinect Hardware, Software, and High-Level Design Alternatives

The Kinect™ device is a peripheral sensor system designed to operate as a motion capture and control input with the Microsoft® Xbox® gaming console. The device has a variety of sensors, including: a video camera, an Infra-Red camera, 4 microphones and a 3-axis accelerometer. The depth sensor works by use of light coding. The device projects a static, pseudorandom and known IR pattern from the IR projector. The device also has an IR camera that detects the return of this IR pattern from the environment. The Kinect™ then compares the IR pattern received to the known pattern that was transmitted and constructs a depth scene [16]. The Kinect™ also contains a variety of processor chips and controllers. The most relevant of these for the current effort is the PrimeSense image processor which preprocesses the 2 camera outputs prior to transmission over the USB interface. This report focuses on the integration of the video and IR (depth) cameras in the Simulink® environment. For the current research, this integration is enabled by the *libfreenect* open-source driver made available on the OpenKinect.org site. This driver provides a simple C interface to the Kinect™ device and makes the video and depth streams available through an API. Additional drivers are available, including one developed by PrimeSense called OpenNI and a recently released driver from Microsoft®. The PrimeSense

drivers offer a C++ oriented abstraction library for depth cameras which, while interesting in its own right, makes it less appropriate for interfacing with Simulink®. The Microsoft® driver was not available at the start of this project, and is intended for Windows-based platforms. The ultimate intention of this project is to use a Linux-based embedded target as the host platform, so the *libfreenect* APi was chosen as the basis for the project.

While the *libfreenect* API provides an invaluable interface to the capabilities of the Kinect™, it is important to consider the environment in which it will be used. Advances in sensing technology are matched, if not outpaced, by advances in low-cost computational hardware. The increase in computational power is strongly correlated to programming complexity, and this has motivated a move away from low-level programming in assembly language or even C, towards higher level programming tools and environments. To varying degrees, these tools allow users first to simulate their designs, then implement them on target hardware (using automatic code generation), and finally to tune system parameters while the code is actually running in real time. Specifically, these tools include Microsoft® Robotics Studio (MSRS), LabView from National Instruments, and MATLAB® / Simulink® from the Mathworks. The MATLAB® / Simulink® environment is arguably the most pervasive within industry and the STEM community, is already tightly integrated into the research activities and educational curriculum at Villanova University and other institutions. Simulink® was therefore chosen as the design environment for the project, and the task was then to develop a seamless bridge between the *libfreenect* API and the user-friendly, block-oriented Simulink® environment.

Figure 1. The Microsoft® Kinect™

3. The VU-Kinect Block

The Villanova University Real Time Kinect block (VU-Kinect) provides a high-level interface to the Kinect™ hardware for Simulink® users, as well as the low-level back-end code necessary to interface to the Freenect API. From a user perspective, the VU-Kinect block appears within the Simulink® block library browser as shown in Fig. 2, and users simply drag and drop the block into their design in order to access the Kinect™ camera and depth images. The block outputs four 480 x 640 pixel streaming signals: one each for red, blue and green from the Kinect™ video camera (hereafter referred to as the combined RGB signal) and one for the depth camera. The RGB signals are unsigned 8-bit integers and the depth signal is an unsigned 16-bit integer. The Kinect video camera has multiple resolution settings which stream at corresponding frame rates. For this research the standard resolution, streaming at approximately 30 frames per second (fps), was implemented. The block can operate in 'Simulation mode' where the user model runs on the host system on a free-wheeling time base, or the design can be transformed automatically into a real time target-system executable operating on a fixed time base. The target system can either be the host machine itself, or some other Linux-based target device such as the Beagleboard or Pandaboard [17].

The details pertaining to real-time embedded applications are worthy of some comment, particularly since a common end application is for autonomous navigation of mobile robots. The process is illustrated in Fig. 3. The start point is a user design in the form of a Simulink® model. When the user initiates a 'build' command, the Real Time Workshop automatically generates the corresponding C code, as well as a 'makefile' which defines how to compile this code into a real-time executable. The VU-Kinect block provides the custom code necessary to interface to the Freenect API. Other Linux target specific code may be required depending on the desired destination hardware platform. The combined code is then automatically compiled and linked into an executable which is either run on the host machine if this is the chosen 'target', or downloaded and run on some other target platform. As shown in Fig. 3, it is also possible to include code for 'External Mode' communications which enable the user to interact with the target during runtime, (as indicated by the dotted line in the figure). This feature is very useful for tuning model parameters during execution and for streaming data back to image viewers in the original Simulink® model to enable the user to monitor the real-time output of the VU-Kinect block or any subsequent transformations of the image stream.

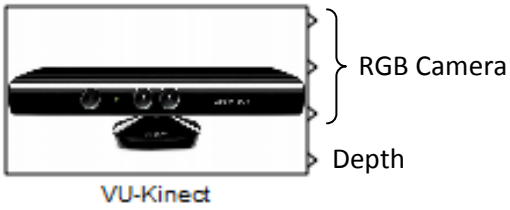

Figure 2. The VU-Kinect Block

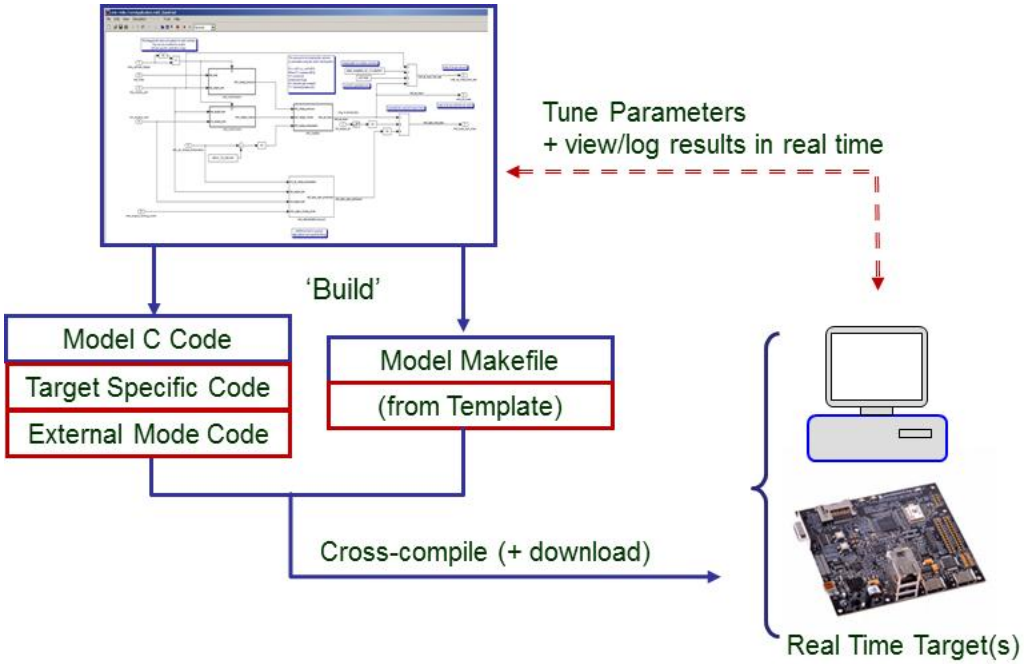

Figure 3. The Rapid Prototyping Process

4. An Object Detection Example

In order to demonstrate the utility of the VU-Kinect Simulink® block and the value of an integrated 3-dimensional video camera solution, a simple object detection demonstration is presented. The objective is to highlight issues with standard image-based object detection techniques and demonstrate the value of augmenting the object detection process with depth information. For the purpose of the demonstration, the VU-Kinect Simulink® block is used in a model with several standard Simulink® Video and Image Processing Blockset™. In the demonstration, a person wearing a shirt with a large (and obnoxious!) print pattern is in the foreground KinectTM field of view (FOV) with another person in the background. The example demonstrates the inability of standard edge detection techniques, applied to the RGB video, to correctly identify the persons as unique objects. However, by combining the information for the depth camera with the RGB video, the persons can clearly be identified.

The experiment was conducted using MATLAB® version R2010a and Simulink® 7.5 on a standard Linux desktop platform (ie, no special computing environment was required). The Video and Image Processing Blockset™, (now called the Computer Vision Blockset™), was used for processing and viewing the video signals generated by the VU-Kinect block. The use of this toolbox is not technically required, but provides a great deal of useful functionality and demonstrates how powerful applications can be constructed with relative ease. The example application, shown in Fig. 4, makes particular use of the Edge Detection block. The application processes the image and displays four different videos for comparison:

1. the raw RGB video image
2. the raw depth output
3. the output of edge detection on the RGB, overlaid on the RGB video
4. the output of edge detection on the depth image, overlaid on the RGB video.

The model starts with the VU-Kinect block in the top left corner. The RGB signals are immediately converted from RGB colorspace into a grayscale intensity signal and then fed into

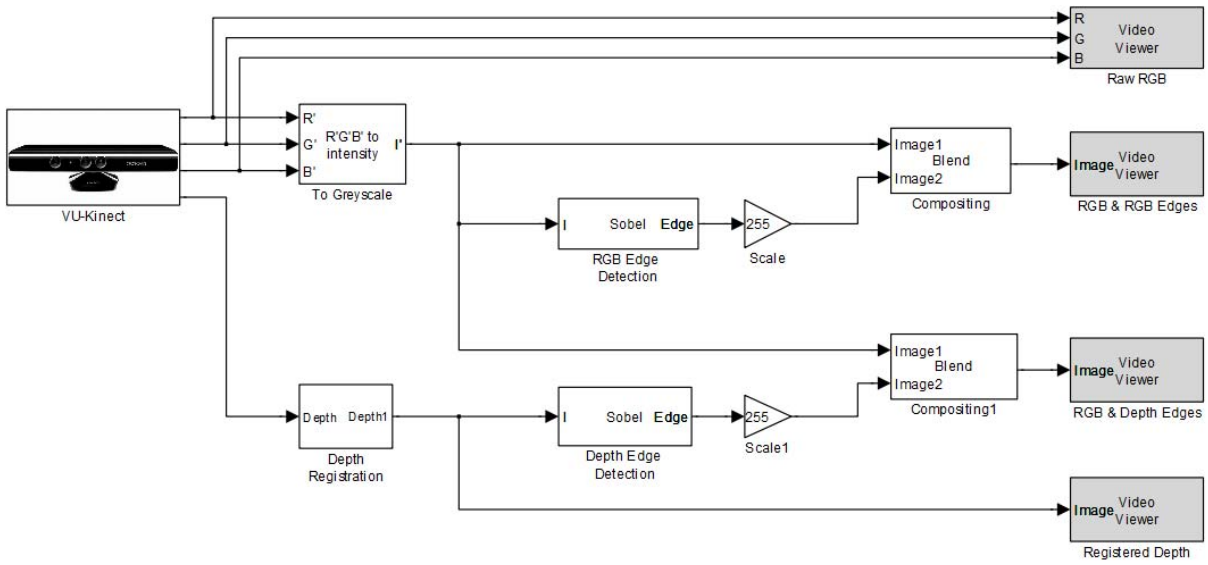

Figure 4. Simulink® Model for Object Detection

the Edge Detection block. For this example, the Edge Detection block is implemented with the default *Sobel* edge detection algorithm which approximates the gradient of the RGB image intensity at each pixel and uses this gradient to estimate where edge are in the image [18]. The output of the Edge Detection block is a binary image with white pixels where edges have been detected and black pixels elsewhere. A gain is applied to this binary signal so that it has the same range as the 8-bit raw RGB video signal. The raw RGB video signal and the binary Edge Detection output are then passed to the Compositing block which overlays the edges on top of the raw RGB video.

An identical process is used for the depth signal with one critical exception. Before the depth signal can be used in conjunction with the RGB signal, the signal (image) must be registered with the RGB signal (image). Since the RGB and depth images come from separate cameras mounted on the Kinect™ sensor, there is an inherent misalignment in the outputs. The registration / alignment of the depth image with the RGB image is a fundamental step in the model and involves the geometric transformation of the coordinate system of the depth camera into the coordinate system of the RGB camera. The specific parameters used for this transformation were developed using the MATLAB® Control Point Selection Tool, which is part of the Image Processing Toolbox™. The tool allows the user to select corresponding points in 2 images and then automatically generates the required transformation to register one image with other.

To demonstrate the importance of the image registration process, the images in Fig. 5 show the unregistered depth image and the registered depth, respectively, overlaid on the RGB video. The image on the left clearly shows the misalignment of the two images prior to registration. Of particular note are the gray 'shadows' around all of the objects on the desk. Once the geometric transformation has been applied, the gray objects in the depth overlay are very closely aligned (registered) with their corresponding objects in the RGB image underneath. This registration process is critical for any applications that intend to make use of both video and depth data.

Once the depth image has been successfully registered to the RGB image, it is converted to unsigned 8-bit integers, and the image is then ready for Edge Detection and Compositing using a process identical to that described above for the RGB image stream.

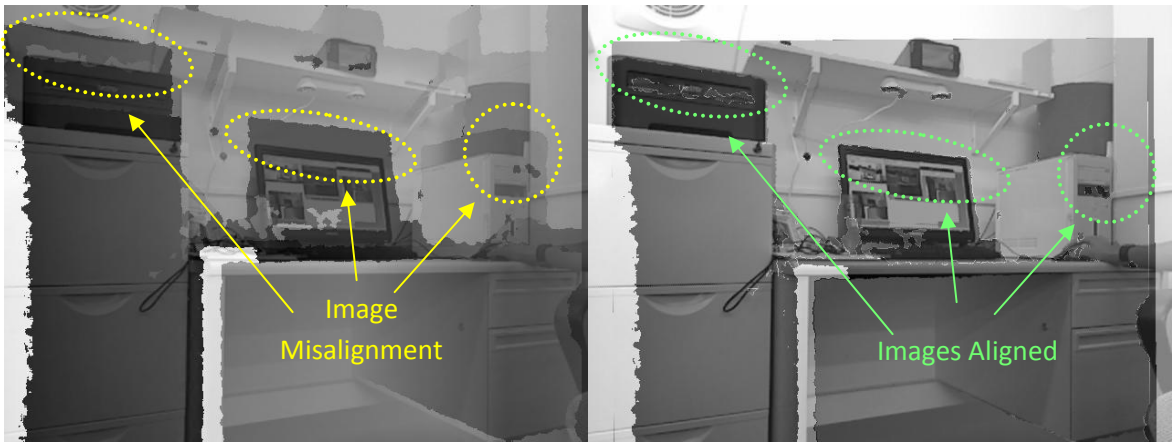

Figure 5. Comparison of Unregistered and Registered Depth Images

4.1 Results & Discussion

Using the model described above, the following results were obtained. The images in Fig. 6 show the raw RGB video output and the registered depth output, respectively. There are clearly two persons in the Kinect™ FOV, overlapping in the plane of the camera, but separated in distance from the camera. By comparison it is easy to observe the limitations in the range resolution of the depth camera. While the main features in the FOV are detected, many of the objects clearly visible in the RGB image are not easily detectable in the depth image (e.g. the phone on the wall).

Figure 6. Kinect™ Color Video and Depth Images

The image in Fig. 7 shows the edges detected in the RGB image by the Edge Detection block, overlaid on the RGB image. The Edge Detection block correctly identifies all of the major features in the FOV: e.g. both persons, the chair on the right, the window and cabinets, etc. However, the block also detects many 'objects' that are not unique objects: e.g. the pineapples and lettering on the shirts of the persons in the image. In addition, there is not clear distinction between the two people in the image (e.g. around their shoulders). From an object detection perspective this would create issues for tracking objects in the FOV, as well as performing other image processing applications.

By using the registered depth image, in conjunction with the RGB image, it is more obvious which edges define unique objects and which do not: The image in Fig. 8 shows the edges detected in the depth image, overlaid on the RGB image. Note that the main features, namely the two persons, are now uniquely and individually defined by their outlines. Of course, the limitation in range resolution does cause the edges detected to be relatively jagged, as compared to the RGB-generated edges. The best processing path forward would depend on the specific application being developed. For the purpose of this example, it is sufficient to demonstrate the differences between the two images.

Figure 7. RGB Image Edge Detection

Figure 8. Depth Image Edge Detection

The intention of the model and examples discussed above is to demonstrate the value of integrating the Kinect™ sensor into the MATLAB® and Simulink® software environment. The VU-Kinect S-Function and Simulink® block allow the Kinect™ sensor data, RGB and depth, to be manipulated and processed by the various tools available in the Simulink® blocksets, as well as by custom developed Embedded MATLAB® Functions. Due to the relatively low cost of the Kinect™ sensor, as compared to other video and depth sensing solutions of comparable resolution, and the availability of drivers and MATLAB® and Simulink® functions and blocks, this development opens the door for many areas of research and development.

5. Conclusions

The VU-Kinect block helps realize the potential of the Kinect™ sensor in teaching and research by providing a seamless pathway from high-level Simulink designs to the low-level *libfreenect* device drivers. In particular, the block provides access to the RGB video and depth images streaming from the device at a 30 frames per second rate. As illustrated in the simple example, it is then possible to draw upon the existing and extensive library of Simulink image and signal processing blocks in order to construct sophisticated applications without needing to program in C, or to engage in low-level programming. These benefits make the exciting capabilities of the Kinect™ much more accessible for non-computer specialists, opening new avenues for teaching and research in signal or image processing, autonomous systems and robotics, and control. Ongoing work is now focused on exploring these opportunities, initially through senior design projects, but with the intention of incorporating the Kinect™ within the experiments of the regular taught curriculum.

Acknowledgements

The authors gratefully acknowledge support for this project from the National Science Foundation (DUE No. 0837637 [16]), and The MathWorks, Inc. MATLAB®, Simulink® and Video and Image Processing Blockset™ are registered trademarks of the The MathWorks, Inc. This work is neither endorsed nor maintained by Microsoft Corporation. Microsoft®, Kinect™ and Xbox® are registered trademarks of Microsoft Corporation.

Bibliography

1. L. Greenwald, and D. Artz, “Teaching artificial intelligence with low cost robots,” In Accessible hands-on artificial intelligence and robotics education, ed. L. Greenwald, Z. Dodds, A. Howard, S. Tejada, and J. Weinberg, pp. 35-41. Technical Report SS-04-01. Menlo Park, CA: AAAI Press, (2004).
2. S. Coradeschi and J. Malec “How to make a challenging AI course enjoyable using the RoboCup soccer simulation system, in RoboCup-98: Robot soccer world cup II: Lecture notes in artificial intelligence, vol. 1604, pp.120-124, ed. M. Asada and H. Kitano. Berlin: Springer, (1999).
3. M. Goldweber, et al. “The use of robots in the undergraduate curriculum: Experience reports,” Panel at 32nd SIGCSE Technical Symposium on Computer Science Education, Charlotte, North Carolina..
4. G. Droge, B. Ferri, and O. Chiu, “Distributed Laboratories: Control System Experiments with LabVIEW and the LEGO NXT Platform,” submitted to the ASEE Annual Conference and Exposition, San Antonio, (June 2012).
5. F. Klassner, K. Lehmer, J.C. Peyton Jones, Genetic Algorithms with LEGO Mindstorms and MATLAB, submitted to the 25th International Florida Artificial Intelligence Research Society Conference (2012).
6. Peyton Jones, J.C., McArthur, C., Young, T. The VU-LEGO Real Time Target: Taking Student Designs to Implementation. Proceedings of the ASEE 2011 Annual Conference & Exposition, June 26-29 2011, Vancouver, Canada. (2011).
7. Vernier Inc, website: http://www.vernier.com/engineering/lego-nxt/
8. MindSensors Inc, website: http://www.mindsensors.com/
9. Camera Reference needed

10. E. Suma, B. Lange, A. Rizzo, D. Krum and M. Bolas, “FAAST: The Flexible Action and Articulated Skeleton Toolkit,” Proceedings - IEEE Virtual Reality, p 247-248, (2011).
11. M. Roccetti and G. Marfia, “Recognizing Intuitive Pre-Defined Gestures for Cultural Specific Interactions: An Image-Based Approach,” 2011 IEEE Consumer Communications and Networking Conference, CCNC'2011, p 172-176, (2011).
12. R. Jota and H. Benko, “Constructing Virtual 3D Models with Physical Building Blocks,” Conference on Human Factors in Computing Systems - Proceedings, p 2173-2178, (2011).
13. Simulink for Natural Integration Device (NID), T. Chikamasa, MathWorks File Exchange File ID #32318, website: www.mathworks.com/matlabcentral/fileexchange
14. Kinect MicroSoft SDK, MathWorks File Exchange File ID #33035, website: www.mathworks.com/matlabcentral/fileexchange
15. Kinect SDK with Matlab, MathWorks File Exchange File ID #32586, website: www.mathworks.com/matlabcentral/fileexchange
16. PrimeSense Ltd., website: www.primesense.com.
17. PandaBoard website: www.pandaboard.org.
18. “A 3x3 Isotropic Gradient Operator for Image Processing”, Pattern Classification and Scene Analysis, 1973
19. J.C. Peyton Jones, F. Klassner, S. Kulkarni, C. Nataraj, “Introducing undergraduates to complex systems through rapid-prototyping of low-cost, networked mobile robots”. National Science Foundation DUE No. 0837637.