

PAPER • OPEN ACCESS

Using LabVIEW Linx for Creating 3d Objects

To cite this article: R Rob and C Panoiu 2019 *IOP Conf. Ser.: Mater. Sci. Eng.* **477** 012027

View the [article online](#) for updates and enhancements.

Using LabVIEW Linx for Creating 3d Objects

R Rob and C Panoiu

Politehnica University of Timisoara, Department of Electrical Engineering and Industrial Informatics, 5 Revolution Street, Hunedoara, 331128, Romania

E-mail: raluca.rob@fih.upt.ro

Abstract. LabVIEW Linx is a LabVIEW add-on for programming applications on external board devices like Arduino, Beagle Bone, Raspberry, etc. Present paper presents a hard-soft application which acquires in real time a 3-axis accelerometer using an Arduino Uno board. The 3d movement generates in real time a virtual object whose moving follows the accelerometer.

1. Introduction

Linx is an open source project by Digilent and is designed to make it easy to develop embedded applications using LabVIEW. LINX includes VIs for over 30 of the most common embedded sensors as well as hardware agnostic APIs for accessing peripherals like digital I/O, analog I/O, PWM, I2C, SPI, and UART.

Whether the user is remotely controlling a chipKIT or Arduino over USB/Serial, Ethernet or Wi-Fi, or deploying VIs to run on BeagleBone Black or Raspberry Pi 2/3, LINX and LabVIEW make it easy visualize the data to work with, debug the code, and create advanced embedded applications faster than ever before.

2. How Linx works

Linx is a hardware abstraction layer that allows user to have a single LabVIEW interface to a variety of different hardware devices. Depending on the device, it can be accessed in one of two ways: Remote or Local I/O.

2.1. Remote I/O subsection. Figures

Remote I/O (Figure 1) works by running LabVIEW VIs on the development PC. In this configuration, the LINX VIs send commands to LINX firmware running on a device connected to the development computer by a USB, WiFi, or Ethernet connection. These commands communicate the firmware to perform pre-defined actions. Devices that use Remote I/O are Arduino, chipKITs, ESP8266s, and many others.



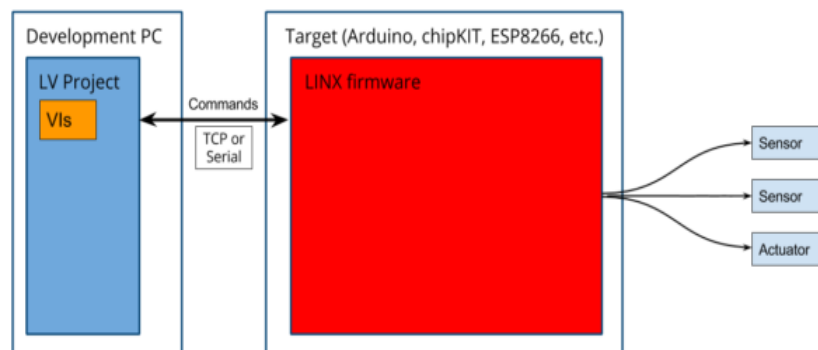


Figure 1. Remote I/O mode

2.2. Local I/O

Local I/O (Figure 2) is used on devices that can execute LabVIEW VIs directly without help from a development PC. Devices that use Local I/O are Linux-based devices like myRIO, Raspberry Pi 2/3, and BeagleBone Black.

In this configuration, VIs are written on the development PC, then deployed and run on the target via a TCP connection with the LabVIEW run-time engine running on the target. While the VIs are running on the target, debugging information and front panel values is transferred between the target and the development PC. When a VI is running on the target with live front panel data on the device PC, this is called Interactive Mode.

The LabVIEW run-time engine on the target runs inside a chroot, which is a Linux construct that is similar to a virtual machine. This allows the LV run-time to be installed and run safely on many different Linux operating systems. LabVIEW VIs running on the target utilize the LINX firmware, compiled as a Linux shared object, in order to access I/O on the target. The LINX shared object contains the logic necessary to access I/O on the specific target board.

2.3. Local I/O with Startup Executable

Local I/O (Figure 3) has a final mode it can be used in via a startup executable. As shown in the diagram above, a development PC is no longer needed in this configuration, since the startup executable is run automatically when the target boots.

The only difference in this mode is that the front panels are no longer visible since the development PC is not controlling the execution of the application's VIs.

This mode is "headless" in the sense that there is not a built-in user interface. Note that although startup executable are called "executable" they do not function like normal Linux programs and cannot be invoked from the command line.

Creation and deployment of the startup application are controlled from the development PC.

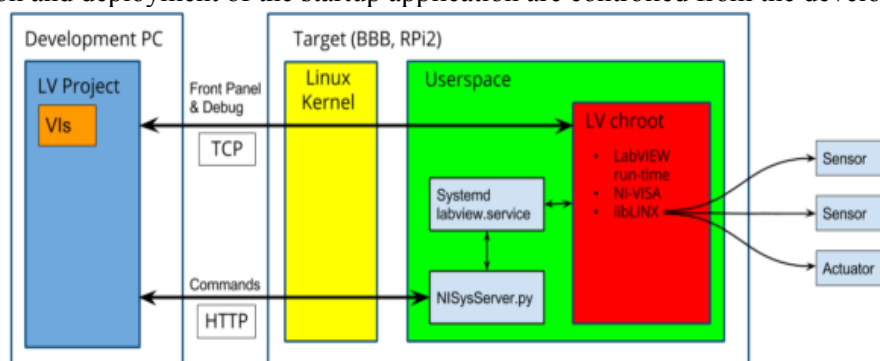


Figure 2. Local I/O mode

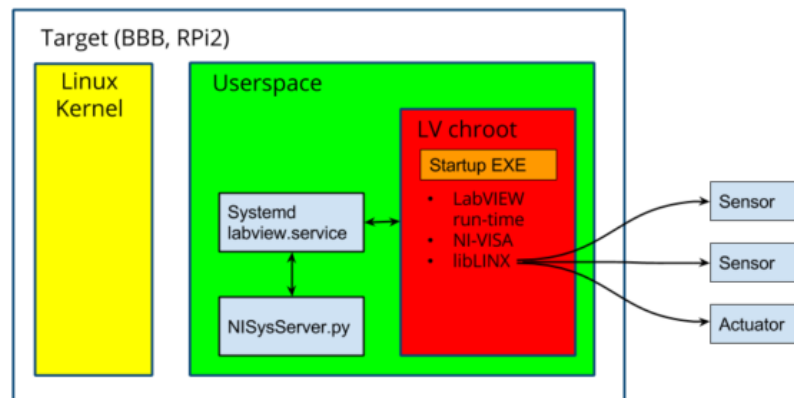


Figure 3. Local I/O with a Startup Executable

3. Using accelerometers in applications (ADXL 345)

Accelerometers are devices that measure acceleration, which is the variation of the velocity of an object. They measure in meters per second squared (m/s^2) or in G-forces (g). A single G-force for us here on planet Earth is equivalent to 9.8 m/s^2 , but this does vary slightly with elevation. The accelerometer movements are presented in Figure 4.

The application which is presented in this paper uses 3-axis ADXL 345 accelerometer with high resolution (13-bit) measurement at up to $\pm 16 \text{ g}$. Digital output data is formatted as 16-bit two's complement and is accessible through either a SPI (3- or 4-wire) or I2C digital interface. The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4 mg/LSB) enables measurement of inclination changes less than 1.0° . In Figures 5a and 5b the ADXL 345 accelerometer is presented.

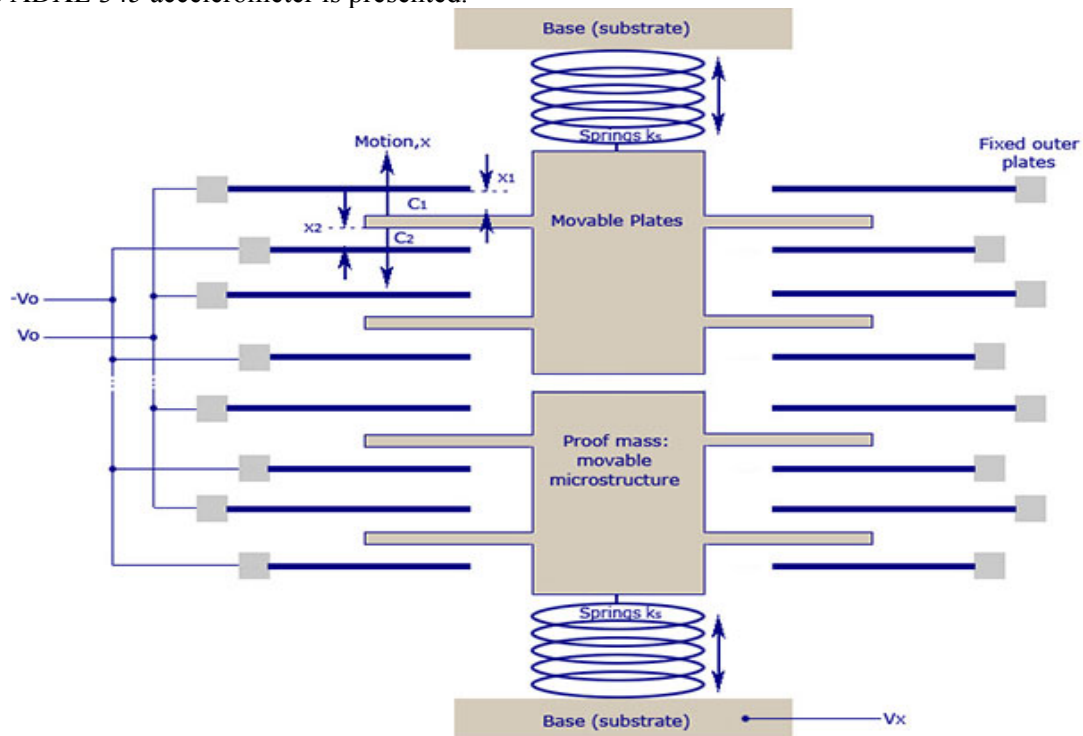


Figure 4. Accelerometer movements

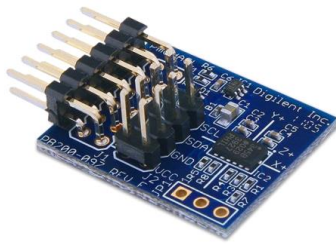


Figure 5a. ADXL 345 accelerometer – Digilent device

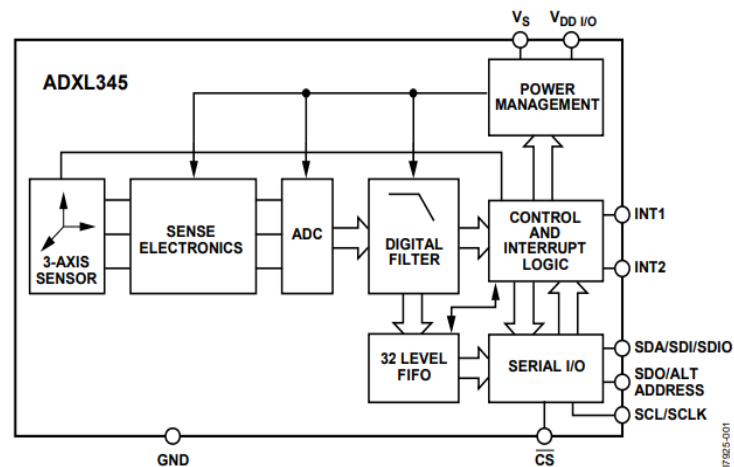


Figure 5b. ADXL 345 accelerometer – Block scheme

4. Application description

Arduino permits to connect the accelerometer to the I2C bus (Figure 6). Using the appropriate firmware, the code can be loaded into the Arduino microcontroller [1]. The code is programmed in LabVIEW Linx. The source code for acquiring accelerometer signals is presented in Figure 7. It contains specified Linx subroutines.

The subroutines which ensure the communication with the devices connected to the I2C bus are the followings:

- Linx_OpenSerial.vi ensures opening the session for serial port communication by specifying the port for loading the program into the microcontroller and also the real time communication with external devices,

- Linx_I2C_Open.vi is presented in Figure 8. It opens the communication session on I2C bus. The slave address of the accelerometer is 1Dh. The I2C channel is 0 by default due to the fact that the Arduino Uno has one I2C channel [2].

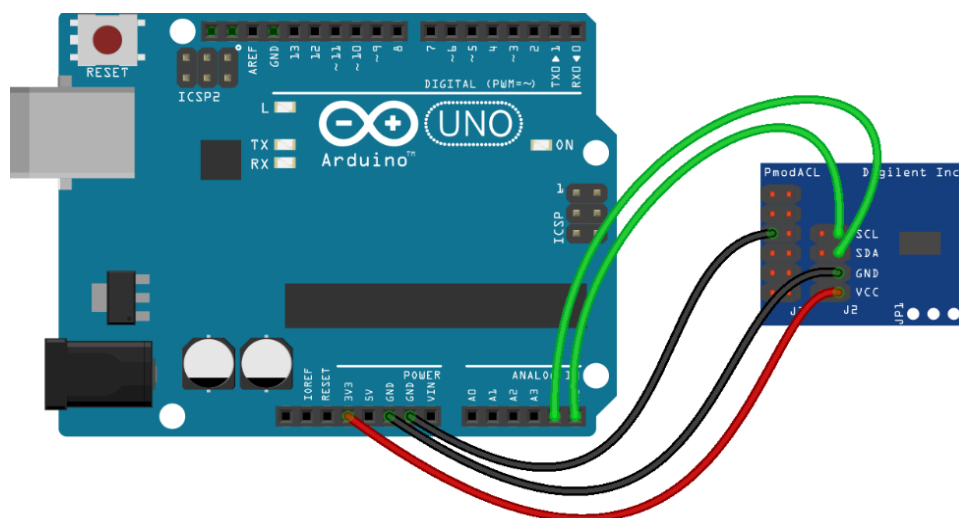


Figure 6. Arduino connection

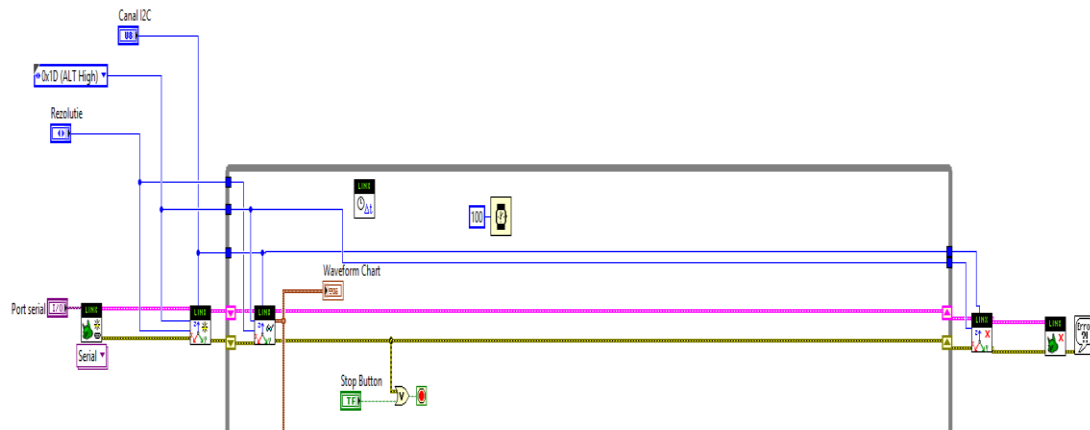


Figure 7. Source code for the accelerometer signals acquisition

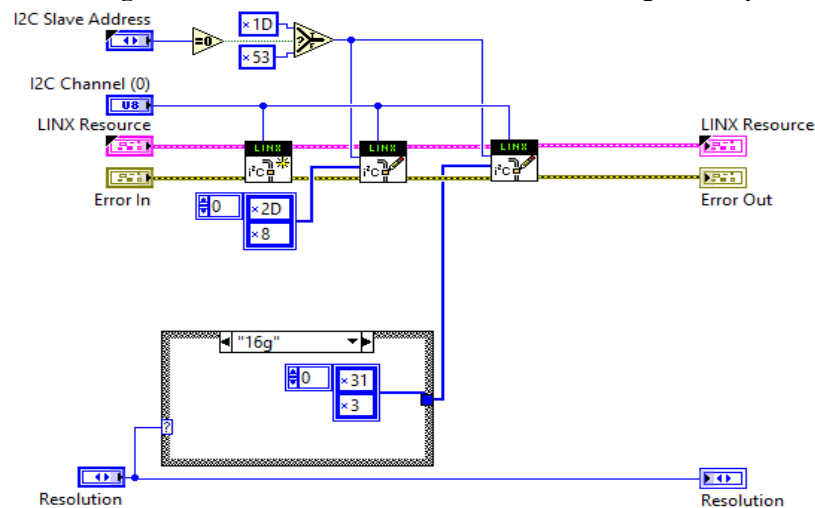


Figure 8. Linx_I2C_Open.vi

-Linx_I2C_Write.vi ensures data transmission at specified addresses [3]. Resolution selection implies using of a case structure with the specified addresses.

-Linx_I2C_Read.vi reads the information provided by accelerometer (Figure 9). This subroutine uses the accelerometer resolution and the used I2C bus.

-Linx_I2C_Close.vi closes I2C session.

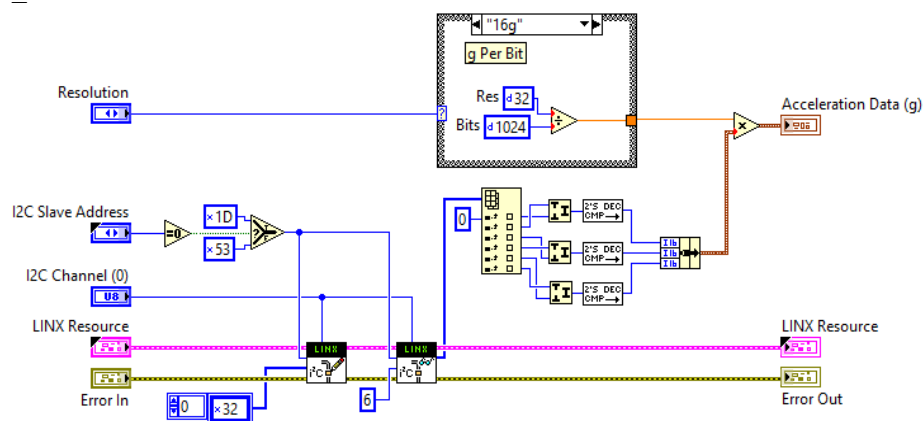


Figure 8. Linx_I2C_Read.vi

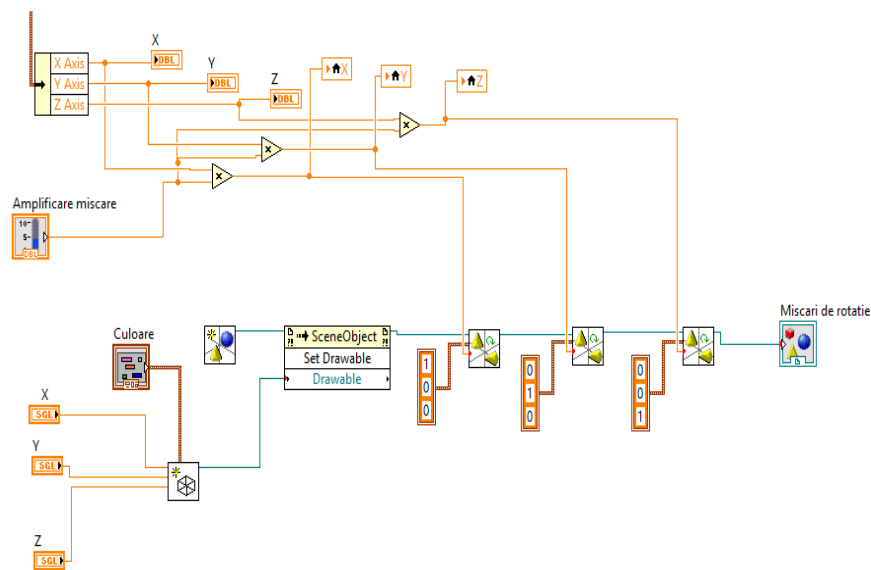


Figure 9. Source code for rotation movement generation

The application designed in this way is compiled and loaded into the microcontroller using Linx Firmware Wizard which offers to the used a wide range of embedded devices that can be connected to LabVIEW Linx.

The application generates 3d cuboid objects with different colour which follows in real time the accelerometer movements and transforms them into rotation and translation movements of the virtual objects. The source code for rotation movement generation is presented in Figure 9. The displacements on three axes of the accelerometer are acquired using three local variables named X, Y and Z. Using the specified subroutines Create_Object.vi and Create_Box.vi, the application generates movements around each axis also using Rotate_Object.vi subroutine. With Property_Node option, these properties are added in series. A case structure is used for generation of the translation movement. The first case generates a synchronous movement of the all cuboid objects (Figure 10a). The second case generates an asynchronous movement of the all cuboid objects (Figure 10b), each one is accomplishing a translation along one axis. Figure 11 presents the 3d object moving in real time with the accelerometer movements [4], [5].

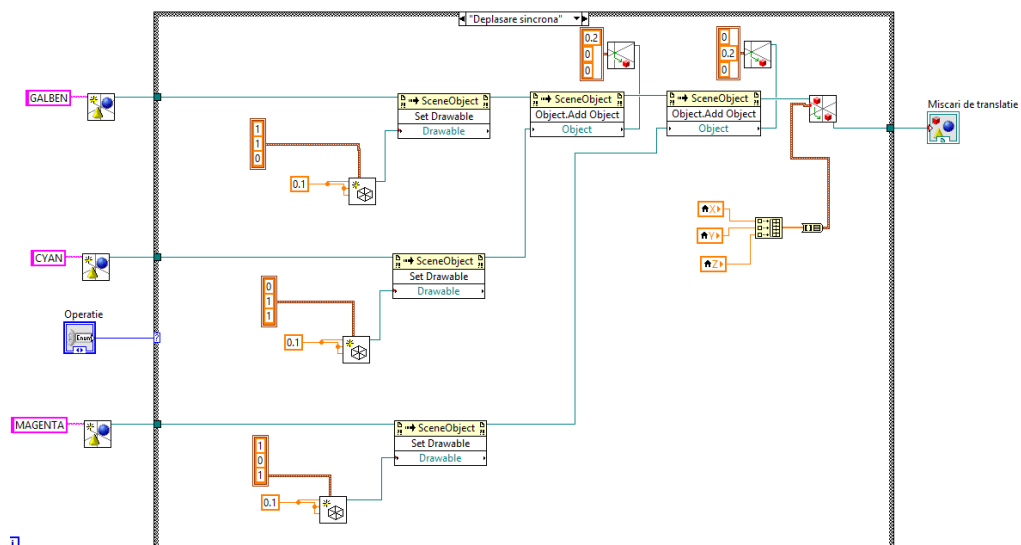


Figure 10a. Source code for generation the synchronous translation movement

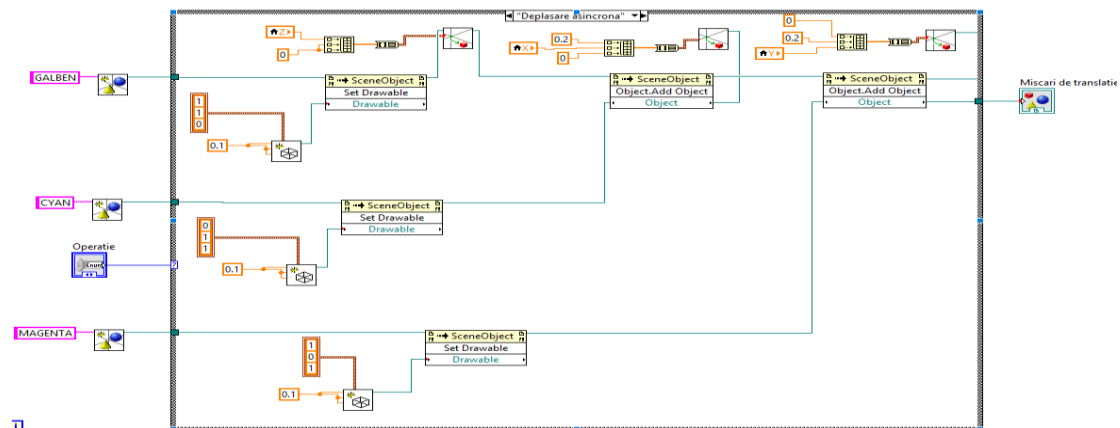


Figure 10b. Source code for generation the asynchronous translation movement

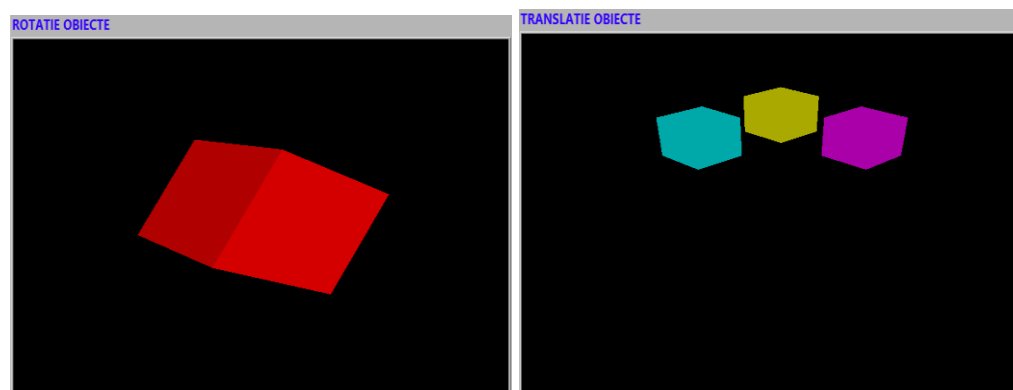


Figure 11. 3D object movement

5. Conclusions

The application presented in this paper presents a new method for controlling in real time external devices using embedded platform with LabVIEW. Arduino board becomes a simple extension board used for acquiring external signals.

References

- [1] Changalasetty S B, Badawi A S and Ghribi W 2014 *Identification and feature extraction of moving vehicles in LabVIEW*, International Conference on Communication and Signal Processing, 3-5 April, Melmaruvathur, Indiapp 1885-1889
- [2] Elya M N, Mohd Noor S B, Ribhan Zafira A R and Azrad S 2017 *Implementation of extended high-gain observer in low-cost otitrack motion system for UAV control*, IEEE 15th Student Conference on Research and Development (SCORED), 13-14 December, Putrajaya, Malaysia, pp 29-34
- [3] Zhen Z, Yang Z, Yuchou L, Youjie Y and Xurui L 2017 *IP camera based LSPIV system for on-line monitoring of river flow* 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI), 20-22 October, Yangzhou, China, pp 357-363
- [4] Dinis C, Popa G N and Iagar A 2017 *Automation heating and pumping for water using Zelio PLC* 10th International Symposium on Advanced Topics in Electrical Engineering, 23-25 March, Bucharest, Romania, pp 545-550
- [5] Cunțan C D, Baciuc I and Osaci M 2015 Operational Study of a Frequency Converter with a Control Sequence Using Xilinx Software, *Acta Polytechnica Hungarica* **12**(6) 201-212