Recent Innovations in Mechatronics (RIiM) Vol 8. (2021) No 1.
DOI: 10.17667/riim.2021.1/5.

Performing Canny Edge Detection Using NI LabVIEW Software Environment

Zalan Tamas Geresi
Mechatronics Department
Faculty of Engineering,
University of Debrecen
Debrecen, Hungary
geresizalan@gmail.com

Syeda Adila Afghan
Mechatronics Department
Faculty of Engineering,
University of Debrecen
Debrecen, Hungary
adila@eng.unideb.hu

Dr. Peter Tamas Szemes
Mechatronics Department
Faculty of Engineering,
University of Debrecen
Debrecen, Hungary
szemespeter@eng.unideb.hu

***Abstract* — Performing image acquisition and processing can require a high amount of computational resource. In this article, I am discussing the application of Canny Edge Detection algorithm to a live webcam image real-time using NI's software environment.**

Keywords — NI, image, acquisition, processing, canny, edge, detection, LabVIEW, webcam, software, development, myRIO, USB, DEMO, machine, vision, module, academic

I. INTRODUCTION

As a result of the collaboration between the Mechatronics Department, Faculty of Engineering, University of Debrecen and NI Hungary Ltd., I was involved in a project, where a system was developed for demonstration purposes.

I had the opportunity to guide the development stages of the system (later referred as DEMO) from scratch, until release. The main goal of this device is to catch the attention of children and youth, inspire new engineering-based ideas and thoughts in them in order to show how tempting it is to become an engineer. The DEMO was born for events where the company represents and advertises itself, such as job fairs and science-based community events.

In a nutshell, when the DEMO is in operation, it creates a live source webcam image and a live edge detected result image on a monitor that is aimed at the crowd in order to catch their attention. When a specific command is given by the user, the system saves the last frame of the edge detected result camera stream as a still image. Shortly, this picture is converted into vector graphical objects, that are drawn to paper using a 2D Plotter. This way, the DEMO is capable of drawing a portrait of a visitor and this picture can be brought home as a memory from the event.

This article focuses on the image acquisition and processing steps that took place in the project from choosing the suitable hardware for the purpose, to collecting the output picture of the software.

II. EDGE DETECTION

A. Edge Detectors in General

Edge detectors play a major role in computer-based image processing and evaluation systems, also in machine vision. The main task is to optimize the resource, required later for image analysis in a way that useful structural information is kept, while the processed data is dramatically reduced. [1]

Its main advantage is the minimalization of failure rate in image analysis. As a result, this technique is often used in object identification and quality assurance.

B. Definition of Edges

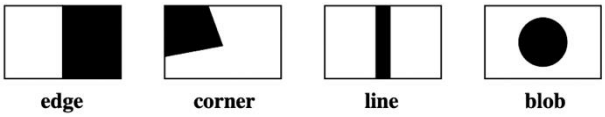

Fig. 1. Types of local image phenomena [2]

Edges are considered to be a high intensity change, perpendicular to the contour. These are the most used local image phenomena in the field of image processing.

In machine vision systems, detection of edges is more typical, since they require significantly less resource than corners, lines or blobs. However, it is important to note that picture edges do not always concur with physical edges. False detected edges can be a result of shadows with intensive contour, reflection or certain textures.

Whenever an edge detector is used, the scene and other circumstances must be manipulated as much as possible in order to minimize false edges (matte surfaces, homogeneous illumination, high applied contrast values etc.).

C. Criteria of Edge Detectors

When comparing different algorithms, the following expectations are examined [2]:

1. Where no physical edge is present, result must be zero.
2. False positives and false negatives must be minimized.
3. Localization must stay accurate.
4. Filter must be independent from edge direction.
5. One edge must be detected once.

D. *Basic Edge Detector Filters*

Each mentioned edge detector is a filter, that applies a convolution mask. This mask is symmetric, often 3x3 size and has an origin in its center element.

During processing, we apply the filter mask to every pixel in the source image using translation. Since the origin of filter is shifted to the inspected pixel and it is required to have exactly eight neighbor pixels, we can not compute values for those, that are located exactly in the border of the image. Finally, the end value is calculated using the corresponding source pixels and the convolution mask. [3]

Roberts Edge Detector

The simplest edge detector filter, where the mask matrix size is only 2x2. Masks are:

$$g_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \; and \; g_y = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \tag{1.}$$

Hence the small size, it requires the least amount of resource. Still, it is the one that is most prone to detect noises. For every pixel, the value is calculated based on its 3 neighbor pixels:

$$G_x = f_{i,j} - f_{i+1,j+1} \quad and \quad G_y = f_{i+1,j} - f_{i,j+1} \tag{2.}$$

The intensity for each pixel is given as the length of the G vector:

$$G = \sqrt{G_x^2 + G_y^2} \tag{3.}$$

Prewitt Edge Detector

The simplest edge detector filter, where the mask matrix size is 3x3. Masks are:

$$g_x = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \; and \; g_y = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \tag{4.}$$

The handling of point-like noises is more efficient in this case then at Roberts. Its drawback is the sensitivity to diagonal edges because the distance between two linear neighbor pixels is shorter than between two diagonally positioned ones. [4]

Sobel Edge Detector

Sobel fixes Prewitt's issue with diagonal edges through weighting the linear ones. Filter masks are:

$$g_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \; and \; g_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{5.}$$

To demonstrate the practicality of Sobel, a sample calculation has been created using Microsoft Excel:

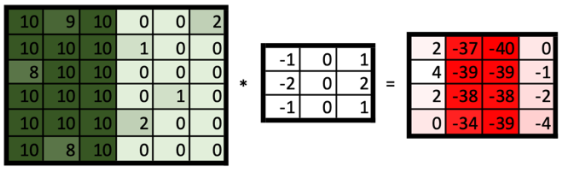

Fig. 2. Sobel Edge Detector, Practical Example [5]

While processing a vertical edge with a slight amount of noise using Sobels' g_x mask, a vertical line is created exactly where the edge on source image is located. Note, that this object has a width of 2 pixels, so post-processing still needs to be done.

III. Canny Edge Detection Algorithm

Canny Edge Detection is most widely used, multi-step edge detector algorithm. Its effectiveness comes from its complexity. While performing conventional edge detection, the additional pre- and post-processing steps are making sure all previously discussed edge detector criteria are met.

It was originally developed by an Australian computer scientist John F. Canny in 1986. Thus, the name originates from him. [1]

A. *Steps*

Application of Gaussian Filter

The main task of the Gaussian Filter is to apply blur to the source image. This way unwanted structural info (in most cases noise) can be eliminated. The convolution mask of the Gaussian filter must have a dimension of (2p+1) * (2p+1) where p is a positive integer. Based on the use case, the size of mask can be 5x5 or 9x9 in most examples.

The amount of intensity reduction between neighbor pixels is directly proportional to their distance. Often a constant is introduced to the formula which is the reciprocal of the sum of the mask elements, to prevent unwanted high values.

$$B_{i,j} = \frac{1}{159} \cdot \begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix} * \begin{pmatrix} f_{i-2,j-2} & f_{i-2,j-1} & f_{i-2,j} & f_{i-2,j+1} & f_{i-2,j+2} \\ f_{i-1,j-2} & f_{i-1,j-1} & f_{i-1,j} & f_{i-1,j+1} & f_{i-1,j+2} \\ f_{i,j-2} & f_{i,j-1} & f_{i,j} & f_{i,j+1} & f_{i,j+2} \\ f_{i+1,j-2} & f_{i+1,j-1} & f_{i+1,j} & f_{i+1,j+1} & f_{i+1,j+2} \\ f_{i+2,j-2} & f_{i+2,j-1} & f_{i+2,j} & f_{i+2,j+1} & f_{i+2,j+2} \end{pmatrix}$$

Fig. 3. Gaussian Filter, Sample Formula [3]

Conventional Edge Detection

Each previously specified basic edge detector filter can be utilized in this step. In general, this filter has a size of 3x3. This can be Prewitt, but Sobel is the most widely used alternative.

Edge Thinning

As seen in *Figure 2.*, the result right after the edge detection contains a two-pixel-wide edge. According to criterium No.3 and No.5 this needs to be corrected. From two edge pixels, the one is kept that has the higher intensity value.

Double Thresholding

After getting the gradient values, we compare them to two pre-defined threshold values called A (low) and B (high) thresholds. This way, pixels are divided into three different sections:

$G_{i,j} \leq A$	Not an edge
$A \leq G_{i,j} \leq B$	Weak edge
$B \leq G_{i,j}$	Strong edge

Table 1. Double Thresholding

Edge Tracking by Hysteresis

Here, weak edges are further inspected. If the weak edge has a strong one as a direct neighbor, it is saved as a real edge, if not, it is aborted. This step is handy, to get rid of false detections based on the locations of edge objects.

B. LabVIEW Implementation

As a part of LabVIEW's add-on, called Vision Development Module, NI offers a near off-the-shelf solution for performing edge detection tasks, using Canny algorithm.

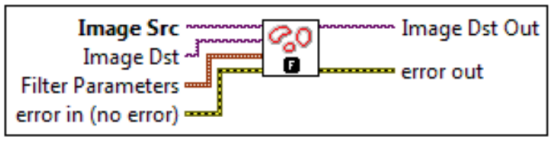

Fig. 4. IMAQ CannyEdgeDetection subVI

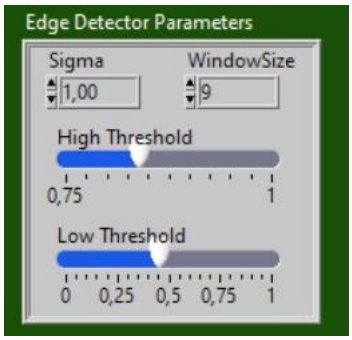

Fig. 5. Filter Parameters Cluster Palette

By giving a source and a destination IMAQ reference session to the subVI, it produces the desired data stream. Filter parameters are shown in *Figure 5.* in detail.

The Gaussian smoothing filter can be set by two values, Sigma and WindowSize (filter size). High and Low Thresholds work exactly the same as discussed above.

IV. Hardware Selection

A. Processing Unit

When choosing the suitable device, which handled the image acquisition & processing, the main requirements were to have the appropriate I/O for connecting a digital camera and the adequate computational resource to perform proper processing. I have compared the devices from NI's academic product palette, since showcasing a DEMO with academic hardware was also a preliminary requirement by the company.

Features	NI myDAQ	NI myRIO (board only)	NI myRIO	NI CompactRIO NI Single-Board RIO
Onboard Processor	–	✓	✓	✓
Linux OS	–	✓	✓	✓ [1]
FPGA	–	✓	✓	✓
Programmable in C/C++	–	✓	✓	✓ [1]
WiFi Connectivity	–	–	✓	–
USB Communication	✓	✓	✓	–
Designed for Students	✓	✓	✓	–
I/O Count	+	++	+++	++++
Onboard Devices (accelerometer, LEDs, push button)	–	✓	✓	–
Onboard Instruments (DMM)	✓	–	–	–
Programmable in LabVIEW	✓	✓	✓	✓
Compatible with NI miniSystems	✓	–	✓	–
Power Supply	USB	Battery/Wall Power	Battery/Wall Power	Battery/Wall Power

Fig. 6. Functionality Comparison of NI's Academic Hardware [6]

The NI myRIO was originally designed for educational purposes on the field of measurement and control, also for basic robot programming. The included FPGA *(Field-Programmable Gate Array)* and CPU *(Central Processing Unit)* work together, enabling the device to run image edge detection tasks at a limited frame rate and resolution. The myRIO has a dedicated USB device port to connect additional peripherals.

Because the NI myDAQ was not able to work as a standalone computing device and the complex modularity of NI CompactRIO was not needed in this task, NI myRIO remained as the best option with its packaged design.

NI myRIO-1900

Fig. 7. NI myRIO-1900 [7]

This NI product is a highly configurable, portable computer with low power consumption. The I/O capabilities include not only dedicated AI, AO, DIO ports, but a 3.5 mm Audio Jack and a USB 2.0 host port. Additionally, its built-in Wi-Fi module supports 802.11 b/g/n standards as well.

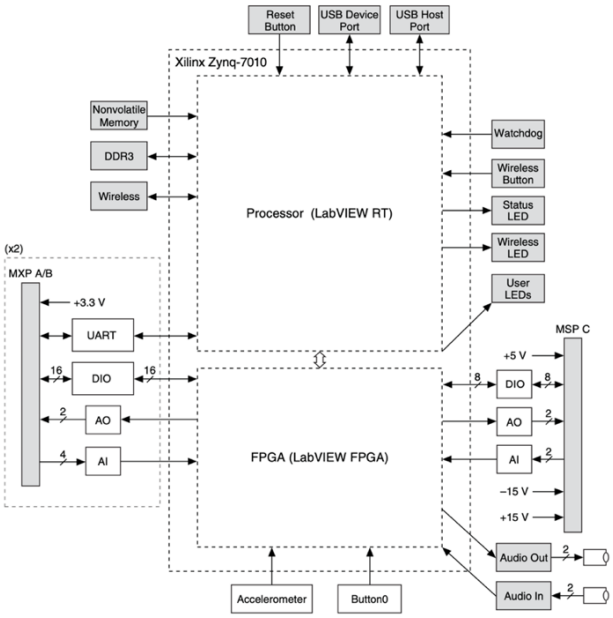

Fig. 8. Structural Diagram of NI myRIO-1900 [8]

The Xilinx Zynq-7010 is technically a hybrid chipset. Its serial processor runs LabVIEW RT, and the included FPGA module runs LabVIEW FPGA. The peripherals are shared according to Figure 8.

The myRIO's included USB Host Port supports three different categories of cameras:

Type	**Description**
Commercially Sold Webcams	All webcams that support UVC (USB Video Device Class) protocol.
Machine Vision Industrial Cameras	Support of USB3 Vision standard and backward compatibility to USB 2.0 is required.
Basler Ace Industrial Cameras	Must support USB3

Table 2. Camera Types supported by NI myRIO [8]

B. Camera Unit

In the current application, with no need for high-end industrial imaging, a UVC supported device was chosen. While keeping costs down, occasional replacement or further replication remains an easier task as well.

Logitech C920

Fig. 9. Logitech C920 mounted, in operation [5]

Since high resolution recording was not taken into consideration, apart from the UVC standard there were no major additional requirements. The mounting option on the bottom and the webcam's high availability in the commercial market were the two advantages why this model was selected.

V. Code Development

Each required functionality, which is in connection with the image processing system was developed in NI's LabVIEW software environment. This way, the code can run both on PC and on myRIO hardware.

A. Initialization

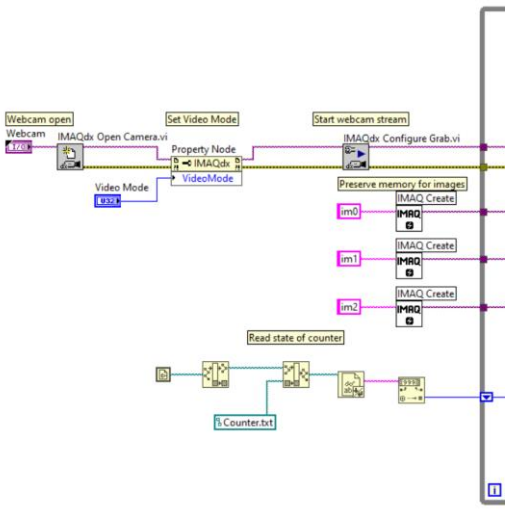

Fig. 10. Initialization Step, Block Diagram

In the first step, communication with webcam is obtained using *IMAQdx Open Camera.vi*, Video Mode is set using a Property Node and webcam stream is initialized. Three different image type variables are defined (im0, im1, im2). Then, the current value of an external file, called *Counter.txt* is read and stored in the shift register of the main while loop.

B. *Image Processing*

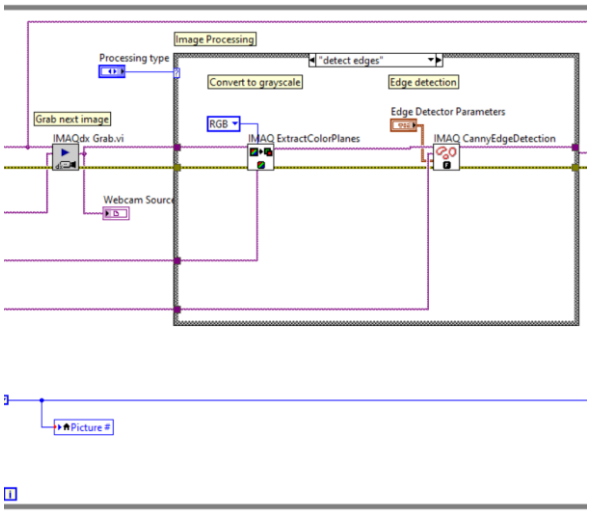

Fig. 11. Image Processing Step, Block Diagram

Inside the main while loop, the webcam source image is stored as im0 using *IMAQdx Grab.vi* and written out to the Front Panel.

Im1 is a grayscale variant of the original stream, processed by *IMAQ ExtractColorPlanes.vi*. Im2 gives the final processed stream after *IMAQ CannyEdgeDetection.vi* is applied.

In the meantime, current picture number (Picture#) is also indicated on the front panel and stored as a local variable in the code.

C. *Capture*

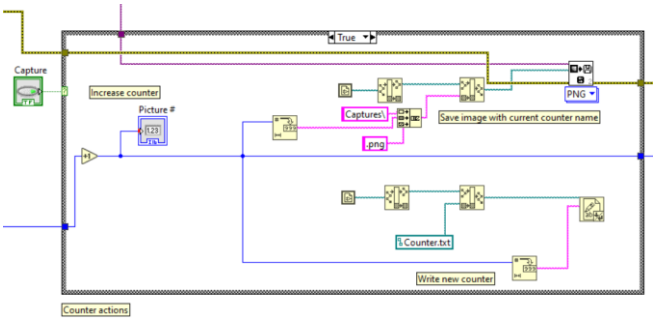

Fig. 12. Capture Step, True Case, Block Diagram

If the *Capture* button is pressed, it activates a Case where the local counter is increased by one, the current frame is saved to an external PNG file with the current counter state as its name and the external *Counter.txt* file's value is also updated.

If no capture action is detected, processed frame is being written out in every iteration.

D. *Delete ALL Function*

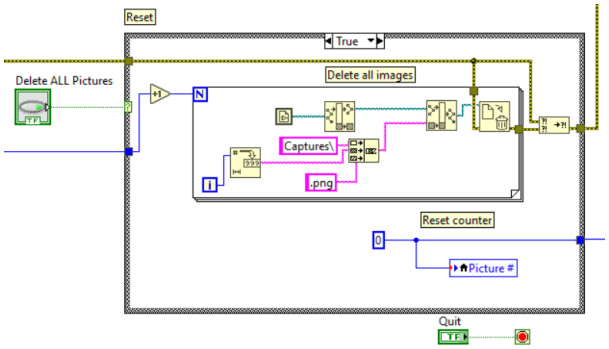

Fig. 13. Delete ALL Function, Block Diagram

The user can reset the program to its default state using the *Delete ALL* function. If operated; the values of the local and external counters are both set to 0. Furthermore, a for cycle runs through, that ensures every previously captured picture is removed from the file location.

E. *Current Framerate Measurement*

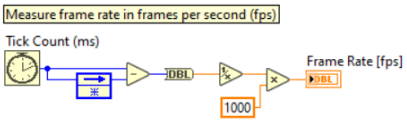

Fig. 14. Current Framerate Measurement, Block Diagram

This minor tool was developed to monitor effectiveness and resource demand. It measures the required time for one iteration, then calculates how many iterations were successfully completed in the last second according to that.

This information is especially useful when choosing the most suitable video mode at the initialization step.

F. *Shut Down VI*

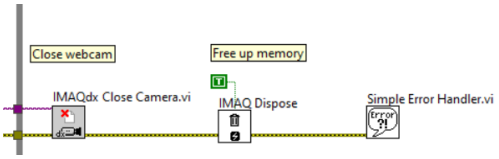

Fig. 15. Shut Down VI, Block Diagram

Finally, *IMAQdx Close Camera.vi* finishes communication with webcam, *IMAQ Dispose.vi* frees up memory by disposing all unused image type variables and *Simple Error Handler.vi* closes the error wire to make debugging easier, if needed.

Fig. 16. Front Panel, User Interface of VI

VI. Validation & Results

The maximum resolution with which the myRIO's hardware could keep the framerate above 12 [fps] was 320x240 pixels. However current framerate is strongly dependent on filter parameters and level of detail in the source picture, this chosen video mode could fully satisfy the needs that were required by this subtask during the DEMO project.

Apparently, personal computers with higher graphical resources could manage to maintain much higher framerates with larger resolutions. While it means an opportunity to further develop the result's quality and stability, it was out of scope for this project due to hardware limitations and already satisfactory results.

VII. Bibliography

[1] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. VIII., no. 6., pp. 679-698, 1986.

[2] D. Csetverikov, Digitális képelemzés alapvető algoritmusai, 2015.

[3] C. K. Somogyi, Féligárnyékos karakterek megtalálása, Budapest: Eötvös Lóránd Tudományegyetem, 2019.

[4] P. S. V., "Continuous Image Acquisition and Edge Detection Using Morphological Filters and Classical Edge Detection Algorithms in Labview," *International Journal of Engineering Research & Technology (IJERT),* vol. VI., no. 2278-0181, pp. 242-245, 2017.

[5] Z. T. Geresi, DrawBOT Demo fejlesztése NI myRIO rendszerrel, Debrecen: Debreceni Egyetem, Műszaki Kar, 2019.

[6] J.-L. Aufranc, "NI myRIO is an Education Platform Powered by Xilinx Zynq-Z7010," CNX Sofware, 6 August 2013. [Online]. Available: https://www.cnx-software.com/2013/08/06/ni-myrio-is-an-education-platform-powered-by-xilinx-zynq-z7010/. [Accessed 19 August 2021].

[7] NI.com, Artist, *myRIO Student Embedded Device.* [Art]. National Instruments, 2016.

[8] I. National, User Guide and Specifications NI myRIO-1900, 2016.