Software solution for Testing Image Processing Algorithm on Conveyor-Based Vision Systems

Software solution for Testing Image Processing Algorithm on Conveyor-Based Vision Systems

Azman Muhamad Yusof[1]*, Ali Yeon Md Shakaff[2] and Saufiah Abdul Rahim[2]

[1] Dept. of Electronic Engineering Technology, Faculty of Engineering Technology, UniMAP.
[2] School of Mechatronic Engineering, UniMAP.

E-mail: *azman@unimap.edu.my

Abstract. Vision systems have been used in many applications that intends to reduce the need for human operators. This is especially true for tasks that are simple but repetitive in nature, which is largely applicable to most manufacturing and agriculture's post-harvest processes. Many such processes utilize conveyor-based systems where the objects being processed are placed on a conveyor belt that runs through multiple processing stations. Implementing a vision system to capture images of an object that is moving usually requires setting up an imaging device to a working conveyor system. Getting a working conveyor system to be ready can take some time and consequently delay development work on the vision system itself, especially those involving image processing algorithms. This paper proposes a software solution that can be used to expedite initial work on such systems. The solution is written in C and is therefore easily ported to any development machine. A basic image processing library has also been developed so that it does not depend on any development library or suite, which is usually huge in size. Thus, the solution can easily be compiled and run on embedded development boards like Raspberry Pi - for a more portable solution.

1. Introduction

Vision systems are getting more attention among researchers and developers. This is understandable from a system point of view, since an image can produce a huge amount of information compared to normal sensing elements. In theory, vision capabilities enable a system to identify objects and their relative positions [1]. Consequently, this also means that vision systems require a more complex processing system with powerful processing elements [2].

Many processing tasks, especially the ones related to manufacturing and agriculture, are now turning to vision systems for simple but repetetive tasks that were traditionally done by humans. As imaging devices and processing elements get faster, real-time image processing applications of many such tasks becomes feasible and realiable for system implementations. Such applications usually involve conveyor systems, where the imaging system is installed at a processing station. Figure 1 shows a block diagram for a generic structure of a conveyor-based vision system.

In manufacturing, vision systems can be used to identify products (i.e. target object) that are being processed either for classification or quality assurance. In [3], such system is being used to detect bottles and to increase the precision of filling the bottles with intended liquid. In another case [4], a vision system is even used to detect defect in on the conveyor belt itself.

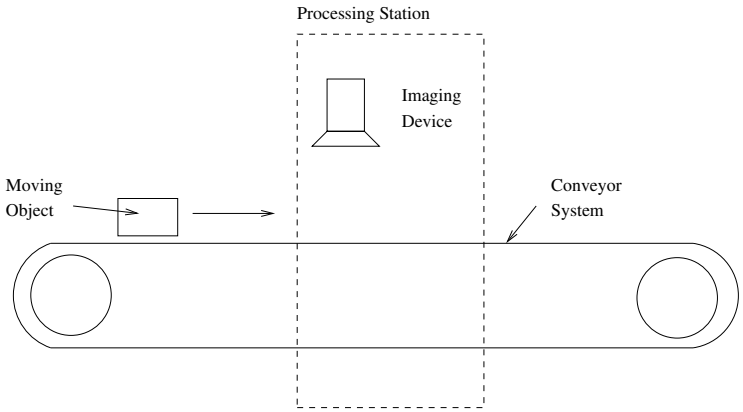

Figure 1. Generic Visualization of a Conveyor-Based Vision System

Agriculture post-harvest processing is another application that is utilizing the usefulness of having vision systems as part of their overall system. Tasks like fruit-sorting [5] and classification (or grading) [6] are actually taking advantage of the availability of such system.

In the next section, a couple of existing implementations of conveyor-based vision systems will be discussed. The following section will cover the proposed software solution for testing image processing algorithms on a conveyor-based system. Consequently, some analysis on practical implementations of the proposed solution will be presented, before more advanced features are discussed in the following section.

2. Existing Work

Most papers do not disclose any information on the methods used at the initial stage of the project when the conveyor is still not ready. Therefore, this section will mostly review a few recent papers that have a conveyor-based vision system setup and analyze the setup of such systems.

2.1. Simple Laboratory Setup

Some systems do not need such tight requirements, either in terms of size or equipments, that it can virtually be built anywhere and still easily moved to another location. This makes it easier to have a mock setup in a laboratory environment, where the researchers can simply start their work.

In [5], a mango sorting system consisting of conveyor system, web camera and a personal computer (PC) has been built. With a dimension of about 1 meter wide, 2 meters in length and height, it can easily be built inside a laboratory environment. Hooking up a web camera and having a desktop computer nearby completes the whole setup. In this work, it seems that all the image processing work was done after the setup was completed, which might have taken a few days if not weeks.

2.2. Industrial-level Setup

Some systems are actually too complex or too tedious to build, that it might as well be built in its intended industrial environment. In [7], such system was built as a novel industrial robot sorting technology. Using industrial manipulator robot arms and intelligent camera, the setup could not possibly be built in a laboratory. The size and setup of the manipulator robot arms themselves would make it impractical to be setup anywhere else. The system uses Cognex Insight 7000 series industrial intelligent camera. Work to set up such a system would have taken

quite a considerable amount of time and would require the software development team to work on site if no simulators are available.

3. Proposed Solution

In order to develop a software solution involving images, a basic image library that defines the image format and provides basic image manipulation functions is required. For that purpose, an open-sourced software *my1imgpro* has been developed and is used as the base for this solution. It is written in C programming language, which makes it highly portable. It has, in fact, been ported to a Xilinx Vertex-II Pro development board, which has a dual PowerPC core running on scarcely-available internal RAM-blocks. This simply shows that it does not require a huge amount of resources to run. The proposed solution involves developing codes that are capable of creating conveyor-like video data, on top of *my1imgpro* library.

3.1. Features of my1imgpro

To represent an image, the *my1imgpro* software uses a data structure named *my1image_t* with 5 members: *width*, *height*, *data*, *length* and *mask*. The first two members, *width* and *height*, are obviously integever values used to store image dimensions. Meanwhile, *data* is a pointer to a dynamically allocated memory of an integer array that holds image pixel values. The size of this pixel array location is saved in *length*. Although *length* seems redundant at first, it can save many multiplication operations required when trying to process the *data* as a one dimensional array.

The *my1imgpro* software was initially developed to process grayscale images. However, as time goes by, it is found that having colour representation is a very useful feature to have. That is the reason for *mask*, the last member in the *my1image_t* image data structure. It can be used to indicate whether the pixel values are actually 8-bit grayscale values or 24-bit RGB (Red-Green-Blue) values. The library also provides functions to manipulate pixels and image regions, functions to load and save image files, along with functions to view the image in Graphical User Interface (GUI). The GUI-related functions are provided by the *GTK* library.

In addition to those mentioned above, a video data structure *my1video_t* has also been defined. It is also a generic data structure, and can theoretically be used with any video capture library and provide a standard interface for applications to analyze the image stream within the video. In order to prove that this structure is adequate, the *ffmpeg* audio-video library has been used as the backend interface to the capture medium, which can be either a file or a live camera feed.

The *my1imgpro* software is available at *https://github.com/azman/my1imgpro* as an open-source software.

3.2. Conveyor-like video library

With a library like *my1impro* already available, work on creating a conveyor-like video generator becomes a lot easier. Instead of have two possible sources for an image stream, a third option has been added whereby the software reads all image files from a designated path and present them as scrolling images that will make any objects in the image look like they are moving on a conveyor system instead. The general idea is as shown in Figure 2.

From implementation point of view, the code basically finds all image files in a path through simple verification process and saves the file names of valid images in a queue list. Then, through iteration, the first two images from the queue are loaded into memory and their file names are reinserted at the back of the queue. This obviously gives an infinite supply of images for the conveyor-like video.

Focusing on code simplicity rather than faster loading time, all operations mentioned earlier are currently implemented in a single thread. As soon as the first image goes out of the intended viewport, the next image file in queue will be loaded into memory and attached to remaining

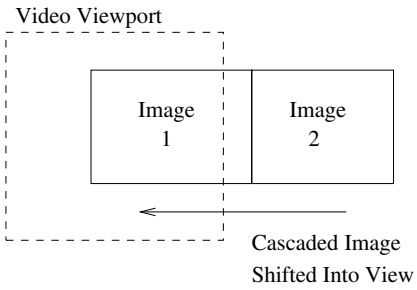

Figure 2. Shifting Image into View for Conveyor-like Effect

image in view. This might create a tolerable slight pause, depending the loading time of the an image.

The main challenge of implementing the conveyor-like image stream is sequencing the image data, i.e. shifting the image, into view. The two-dimensional image data is actually stored in a single-dimensional array, starting with the first row going into the next. Clearly, accessing the rows is not a problem because pixels in a row are already in sequence. However, accessing a column, which is what we need to do in our case, requires an alternative strategy.

While shifting into view, a variable is used to keep track of the next column available. Using this value as an base index, the column pixel data are obtained by offsetting the index by image width, which would get us to the next pixel in the column. Obviously, the process increases processing time but that can be tolerated at the moment since the code implementation is simpler this way and the output are still acceptable. Despite all this, the application using this code so far has shown sufficiently satisfying results.

The code for conveyor-like video generator will also be released as open-source software in the near future.

4. Implementation Analysis

The proposed solution has been used in a project meant for fruit sorting application. At the initial stage, building a conveyor system can take some time and getting fruit samples for testing algorithms can be a challenge, especially for seasonal fruits. With the proposed solution, having just 2 sample pictures of the target fruit would be enough to get things started. As can be seen in Figure 3, the target fruit for that project are mangoes.

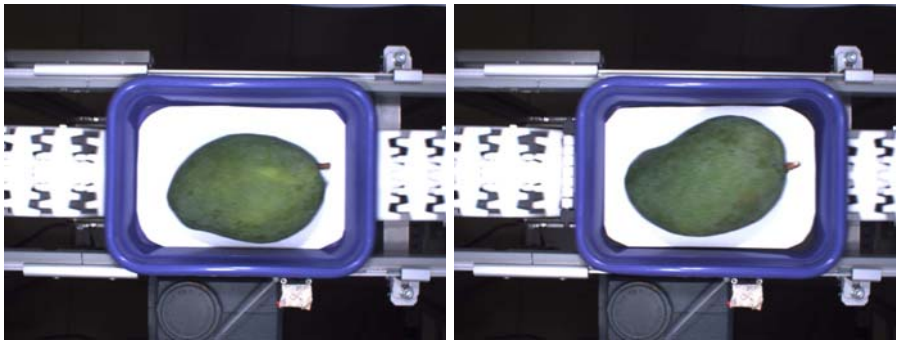

Figure 3. Two Sample Pictures of Mangoes on Conveyor System

In the mentioned project, two pictures of mango which incidently are taken while on a conveyor system, were used as source images. This may not seem much, but with some image

manipulation like rotation using 90 degrees steps and mirroring, we can easily get 16 samples instead. However, even in its original form, the two images can provide endless video image stream for any algorithm testing.

Figure 4 shows how the two images can be used to generate a conveyor-like video. The upper row shows the generated video image stream, while the lower row shows filtered video image stream. As can be seen from there, the filter actually detects mangoes only when it can be seen as a whole mango.

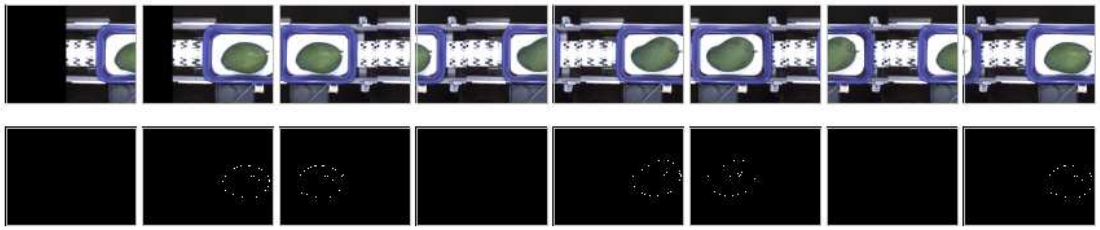

Figure 4. Top row shows the original stream and the bottom row shows filtered stream

Practically, the proposed solution can be used in two different setups. First, a target application with the algorithms being tested can be built together with the conveyor-like video generator code into a single binary. Basically, that means that it can be run together on the same platform (most probably a desktop computer). This is exactly what have been done in the sample demo presented above.

In another setup, the conveyor-like video generator code can be run on a desktop computer and displayed on standard monitor. Meanwhile, the target application with the algorithms being tested can be built separately on the actual intended target platform that has access to an imaging device that points to the monitor. This setup should be more similar to the intended application and therefore provide a more accurate test results compared to the first setup. Figure 5 shows the video generator code running on a desktop PC, while the application code was running on a Raspberry Pi 3 board with 7" display.

5. Conclusion

A software solution to test image processing algorithms on conveyor-based vision system has been presented. It would be very useful in purely research environment where building a conveyor-based system is time-consuming or maybe even not practical. Even without a conveyor system, development work related to vision systems in general or specifically the image processing algorithms can be initiated.

Being written in C programming language makes the proposed solution portable to any platform that has a C compiler - which means, almost all platforms. Since the *my1imgpro* library does not require any huge dependencies, other than *GTK* (for GUI display) and *ffmpeg* (for interface to imaging device and video files), the proposed solution is easy to build. It is also especially useful in embedded systems implementation. The core video generator code can easily be built on Raspberry Pi for a portable vision system solution.

The effect of loading delay of an image when running the conveyor-like video can be reduced by introducing multi-thread implementation. A separate thread can be created to handle loading of the third image while running the first two images. This should create a smoother flow for the conveyor-like video.

Other than the objects on a conveyor system, the conveyor system itself sometimes requires similar setup where it needs to inspect the conveyor belts for any damage periodically. This

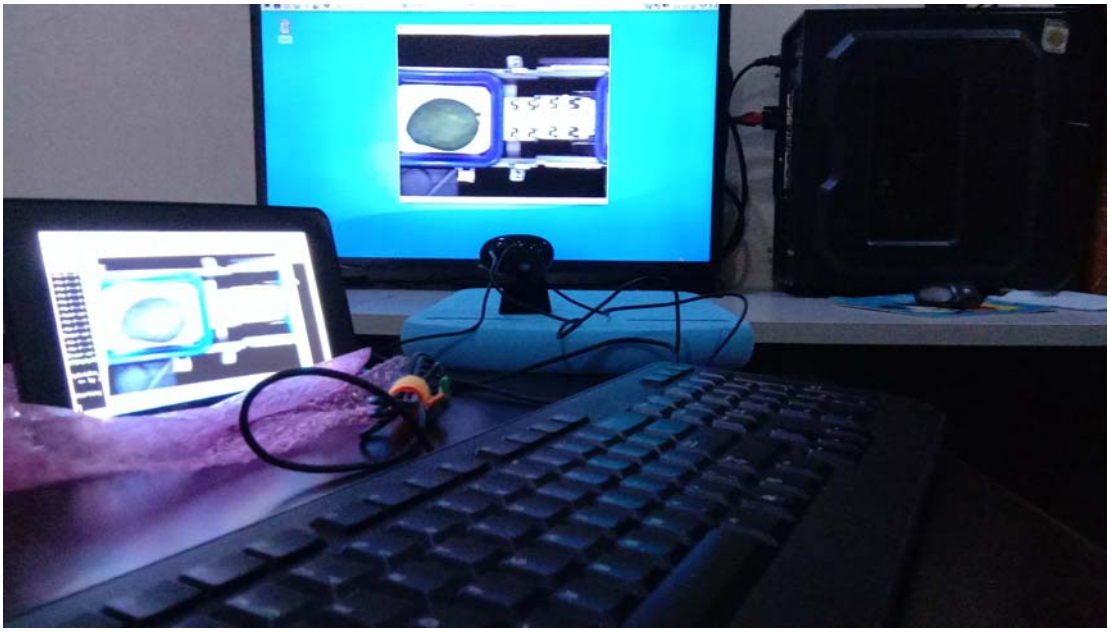

Figure 5. Running image capture and video generator on separate platforms

solution can help in the development of such algorithms without having to have actual damage on the conveyor belts.

Looking further, any applications that involves a vision system looking at objects moving within a certain path could be benefitted by the proposed solution. For example, it can be applied to biomedical analysis tool that is used for blood stream imaging, which is similar to conveyor-like flow.

References

[1] Marr D 1982 *Vision : A Computational Investigation into the Human Representation and Processing of Visual Information* (W.H. Freeman, San Francisco) ISBN 0716712849 0716715678
[2] Yusof A M, Shakaff A Y M and Rahim S A 2018 *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)* **10** 123–129
[3] Thiyagarajan K, Meenakshi R and Suganya P 2016 *2016 International Conference on Advances in Human Machine Interaction (HMI)* pp 1–3
[4] Yang Y, Zhao Y, Miao C and Wang L 2016 *2016 IEEE International Conference on Signal and Image Processing (ICSIP)* pp 315–318
[5] Rojas-Cid J D, Prez-Bailn W, Rosas-Arias L, Romn-Ocampo D B and Lpez-Tello J A 2018 *2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)* pp 1–6 ISSN 2573-0770
[6] Su Q, Kondo N, Li M, Sun H, Riza D F A and Habaragamuwa H 2018 *Computers and Electronics in Agriculture* **152** 261 – 268 ISSN 0168-1699
[7] Li C, Ma Y, Wang S and Qiao F 2017 *2017 9th International Conference on Modelling, Identification and Control (ICMIC)* pp 902–907