TAF

Journal of Applied and Physical Sciences

JAPS

2021, **7**(1): 1-10

PRIMARY RESEARCH

Industrial user interface software design for visual python AI applications using embedded linux based systems

Ulas Dikme *

Department of Electrical & Electronics Engineering, University of Gaziantep, Gaziantep, Turkey

Keywords

Python
C++
Face detection
User interface
Artificial intelligence
Industrial software design

Received: 12 November 2020
Accepted: 1 January 2021
Published: 12 March 2021

Abstract

Python software is one of the most popular languages for artificial intelligence applications, especially in the academic area, because of its easy syntax and ready-to-use libraries. But for industrial usage, due to the nature of the declarative languages, python is not the preferred language when it is needed to add more functionality to the application, like user interactions or other software abstractions, which needs more system resources and stability, especially in embedded systems. If there is not enough resource to build a new model for AI application for desired software language, it will be perfect to have the advantage of the ability of python in the AI field. Instead of creating one python application or more than one python layer in the system, it is efficient to abstract the AI application, which is written in python language, and handle all other activities with more efficient languages or frameworks. In this paper, we will see how we can use a visual python AI application, which communicates with another software layer written by C++ using the Qt framework for a user interface in an efficient way, running in the backend to handle only AI-related processes. In the example, the python application detects faces in the backend and sends related visual data to the frontend application using interprocess communication. The frontend application will be efficient from a memory usage perspective and flexible for customer usage in an industrial way. The whole working demo, consisting of a python face detection application and a C++ program, is available in the given GitHub link [1] and is explained in a detailed way for software design and the user interface, which will be written in QML language.

I. INTRODUCTION

If a software application is industrial, it means that the application must run stable and efficiently from a memory perspective. From the user's perspective, the interaction should be flawless without freezing or slowing behavior as much as possible. With declarative languages, achieving this is quite hard and not necessary. Choosing simple languages that are close to hardware than declarative ones will solve the efficiency and stability issues. Nevertheless, sometimes it is impossible to find ready-to-use AI models or libraries with less abstracted languages according to the hardware. As is seen, for AI development, python is the popular language.

It is ideal to use python only for AI applications and create other layers in the system with more efficient languages or frameworks. In this portable design with AI application, interprocess communication has crucial importance. All layers should be designed to suppress inefficient hardware resource usage of AI applications. This will save time for companies which has not to have much time for development or resource as people. The system also should be abstracted and portable so that in the future, layers can be replaceable with others easily. For instance, for the first time in development, python can be used, and later it can be replaced with another language. Otherwise, the development team will lose time and money.

AI development is not the main focus here. AI application will only work in the backend as a command line pro-

*Corresponding author: Ulas Dikme
†email: ulasdikme@gmail.com

cess. All other programs will be written in C++ using the Qt framework, especially with functional QML modules. In the demo intentionally, a simple python AI application, which consumes more memory resources, is chosen. It is easy to replace it with another python program easily. The demo is running on Linux based operating system.

Especially for graphical operations, drawing primitives to screen with QML [2] will be more efficient when all calculations are done with C++ and sent to QML Qt [3] communication mechanism. Also, the Qt framework will give more flexibility in choosing the graphical pipeline in the operating system. In embedded systems, graphical operations can use noticeable memory if the correct structure is not determined.

A. Significance and Objectives of the Study

Most visual AI research is done with python software, especially nowadays, because of its easy syntax and ready-to-use libraries. But when visual AI research is wanted to be used in real life, especially in embedded systems [4] for industrial usage, there can be hardware problems when adding more functionality to the application, like the user interface or database access. Because of the essence of declarative languages like python, it will consume more CPU power, which means it is not a preferable option for embedded devices. If it is wanted to be used hardware-friendly languages, there may be more time needed to replace the python language for the AI model, which is not practical.

This paper discusses how it is possible to use a visual AI model [5] with python without consuming more hardware resources. To do that, python handles only visual AI-related processes. Hardware-friendly languages and frameworks handle all other software processes like the user interface or hardware access. Related software design is discussed from a system and object-oriented perspective.

II. OVERVIEW QT FRAMEWORK

Qt [6] is a cross-platform development framework [3]that includes graphical user interface modules and C++ libraries. For the modules, it is possible to create efficient 3D designs for simple HMI structures using OpenGL with the scripting language. Also, with external C++ libraries, it is possible to combine the desired functionality with Qt [7]. And also, choosing the desired C++ version, it is possible to add other items or bind external libraries. One of the best features of Qt is the signal, and slot [8, 9] mechanism, which connects two or more objects or functions regardless of their state and dependency. This allows us to easily talk about the graphical design part from the C++ side. In this manner, reasonable to calculate or process algorithms with C++ and represent the result to the QML [10].

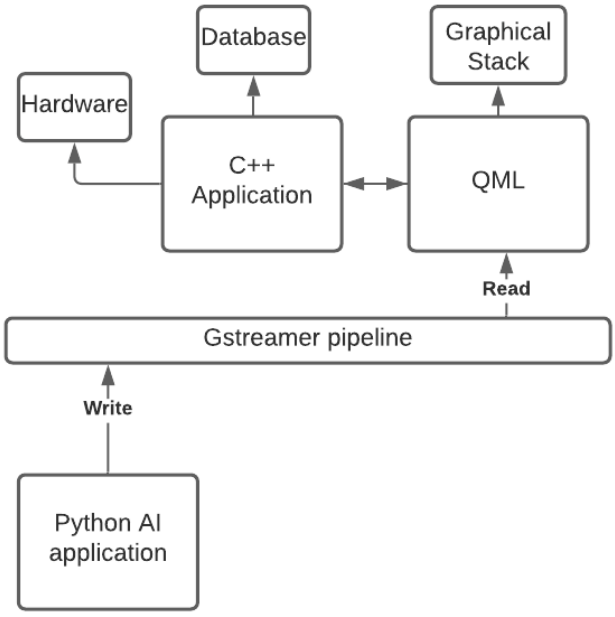

Fig. 1. Block diagram of the application

A. QML language

QML is a declarative language [2] that uses visual modules or components like buttons or 3D fluid animations. The syntax is readable and can be combined with javascript. Usage is straightforward and portable. It is possible to run the same QML [11] application on different hardware and oper-

ISSN: 2414-3103
DOI: 10.20474/japs-7.1

ating systems like windows and Linux. With other modules like multimedia module that benefits the visual part, it will be effortless to create and combine different behaviors. In Listing 1, you can see a simple example.

```
import QtQuick 2.12

Rectangle {
   width: 120
   height: 100
   color: "blue"
   Text {
   anchors.centerIn: parent
   text: "fisrt rect"
   }
   TapHandler {
   onTapped: parent.color = "yellow"
   }
   Rectangle {
      width: 120
      height: 100
      color: "yellow"
      anchors.left: parent.right;
      Text {
      anchors.centerIn: parent
      text: "second rect"
      }
   }
}
```

Fig. 2. QML example

It is possible to create a coordinational relationship between QML objects [11] and easily handle fundamental user interactions like a keyboard, touch screen, or mouse.
To design the visual menus or navigation systems, QML provides model structures that will help to organize the related objects on the screen. In this example, it will be used the stackView [12] model will create a stack to hold each menu element and control it accordingly to the user interactions for the gridView model to produce handy visual interfaces.

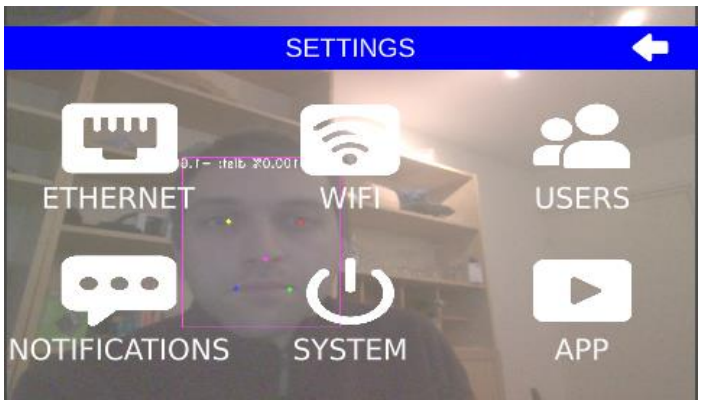

Fig. 3. Menu design with stackview module

ISSN: 2414-3103
DOI: 10.20474/japs-7.1

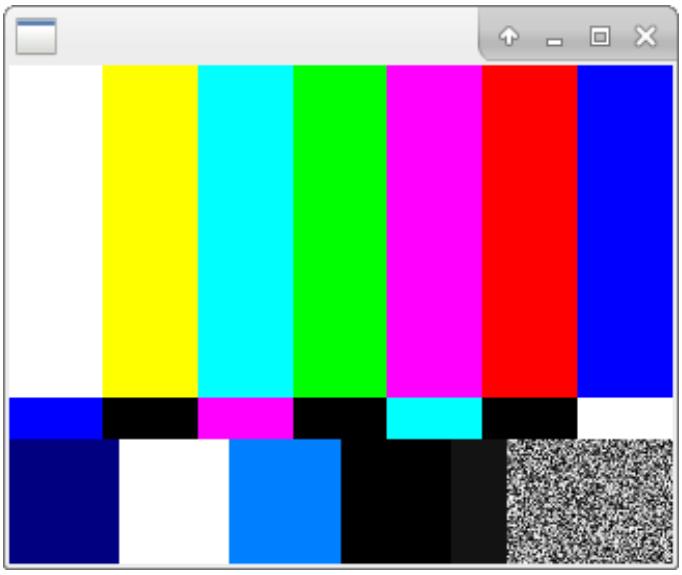

Fig. 4. Output of the Gstreamer pipeline

B. *Gstreamer and Qt Multimedia*

The IPC mechanism, which is handled by the Gstreamer tool, is used to share the frames [13] from AI applications with UI. Gstreamer is a tool [14] to create a pipeline for graphical operations. It can be used directly from the code base or as a command line tool. Using Gstreamer, an application can access the camera, shared memory, or socket to get the frames or related data. It is not only used for video processing. It can also be used for audio applications.
With the command line tool, it is easy to create the pipeline. As seen in Figure 5, the gst-launch is the application name, and there are two more modules. Videotestsrc is the test source for video, and ximagesink is the sink point for the layer that will be used to represent the video. The ximagesink is used for Xserver-based systems.

```
gst-launch-1.0 videotestsrc ! ximagesink
```

Fig. 5. Gstreamer pipeline example

The output of Listing 2 can be seen in Figure 4. There is a source element, which can be a camera or another pipeline output. And there is a sink element to clarify where data may be sent. The created pipeline can be distributed or divided according to needed behavior.
In typical pipelines, there will also be a filter element. For example, to use a camera element, for example, a camera element, sometimes it is necessary to filter it to align the data type with the sink element. A simple pipeline diagram can be seen in Figure 5.

Fig. 6. Simple Gstreamer pipeline diagram

The Gstreamer tool connects the python AI application and the frontend Qt application for the IPC mechanism. Data will be shared over UDS (UNIX domain socket) [15] socket. Usage can be seen in Figure 1 as an IPC mechanism.
It is possible to use the Gstreamer tool with QML directly with the help of the multimedia module. Example usage is in Figure 7. As it is seen for Qt, there is another sink element. For the python visual AI application OpenCV is the interface for Gstreamer [13].

ISSN: 2414-3103
DOI: 10.20474/japs-7.1

```
MediaPlayer {
   id: player
   source: "gst-pipeline:
   nvarguscamerasrc !
   autovideoconvert ! qtvideosink"
   autoPlay: true
}
```

Fig. 7. Multimedia pipeline example

C. Graphical structure and Qt

Windowing systems [16] cannot be efficient with platforms where there are not enough hardware resources, like embedded environments. With Qt, it is possible to choose and configure the graphical stack for the application. It can use Wayland or Xorg [17] or directly access Opengl. For embedded applications, It is recommended to use EGLFS, which is an interface with Opengl. These platforms should be configured during the compilation.

With EGLFS [18], the Qt application will not use Xorg. Because of that, there will be high CPU usage saves. Therefore EGLFS will push the application for full screen. This should also be considered If there is more than one application that needs windowing representation.

Qt framework (for Qt5) creates a GUI structure [19], including graphical elements based on the core. These include non-GUI functions, and QPA (Qt Platform Abstraction) [20] is implemented as a plug-in abstraction layer of the platform basic window system and is dynamically loaded when Qt Application is started.

Efficiency is related to compositor [21]. With EGLFS [22] graphical output is not needed to send to a compositor like Xserver, which brings efficiency. The simple stack can be seen in Figure 12.

III. SOFTWARE DESIGN AND IMPLEMENTATION WITH USER INTERFACE

As it is seen Python application is running in the background. It sends the frames, including AI-processed results, to the Qt application via the Gstreamer pipeline using UDS. Using the multimedia QML module, these frames will be represented on the screen, and the Qt application will draw other graphical primitives like buttons [23], menu objects, title bars, etc. The result can be seen in Figure 4.

A. Design with class diagram

As is seen in Figure 11, there is a class named Network, which represents the network configuration menu element in the graphical design. For each menu element, it is a good idea to use a software design like this in the figure. In QML, it is possible to create modules; in this design, each module has its class used for communication with C++ and the frontend part. When the user clicks the network menu item, Network UI's constructor is called, and when the user wants to leave the network page, Network UI class is destructed. This approach will reduce memory usage because the user does not use all menu items simultaneously.

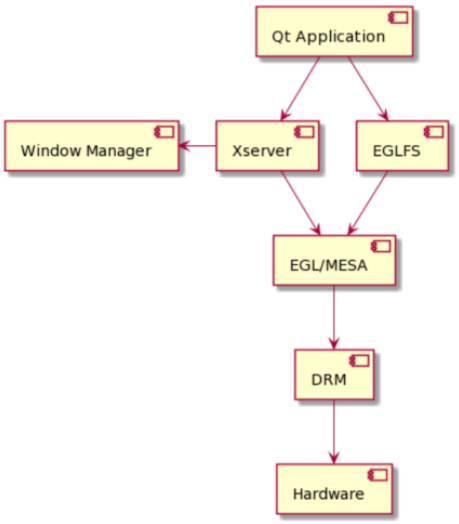

Fig. 8. Simple Linux graphical stack with EGLFS

ISSN: 2414-3103
DOI: 10.20474/japs-7.1

Whenever the user opens a new page from the menu, there are still objects like the title bar or keyboard in the background. These should be able to communicate with the menu objects. These common objects, like the keyboard or title bar, have the same life span as the application. For example, during the network configuration, which can take some time because of the hardware, the user can be updated accordingly using a popup screen or changing the text on the title bar. And also, when the user clicks the text input, the keyboard should appear; therefore, it should not overlap the text input. Otherwise, it is not possible to see what is typed, especially on small screens, so when the keyboard appears, text input should go up according to the height of the keyboard. The size information of the keyboard can be held in the abstracted object. As is seen in Figure 6, the network menu item is open, and there is no keyboard, but when the user wants to edit one of the text inputs, the keyboard will pop up from the downside of the screen. The problem is that it should not overlap the text input. Otherwise, it is impossible to see what is entered or updated by the user. This issue can frequently happen on the small screens used in embedded applications. In figure 10, you can see that the keyboard is active, and text input areas are moved to the upside according to the keyboard's width. Therefore, the keyboard object does not belong to the menu class; otherwise, it should be constructed when a menu item is created. This can create slow behavior during each user interaction. That is why the keyboard should be abstracted from the menu items and related data held by the different classes. Also, data can be used commonly from all menu items. That is why it is a good idea to create an object at the beginning of the program which holds object information and a pointer to access. And each menu item can use them to access it or to write it. There can be many objects. In the design, there are two classes: one is for the screen keyboard, and the other is for the title bar to show notifications to the user in case of any situations. One adapter interface can return the desired pointer of the controller class to menu objects. This reduces dependency on a class named Network UI.

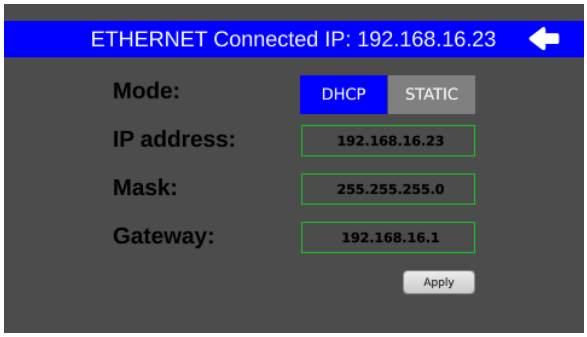

Fig. 9. Network page without keyboard

Fig. 10. Keyboard object does not overlap the network page items

An interface may handle network configuration, such as IP address update or representation. These operations can take some time because of hardware dependency. Users should be updated about the current situation during this time, and this page should still stay alive. And also, during the configuration time user interface should still be active, and there should be no freeze or slow behavior. To achieve this, throwing a thread is a simple idea to run the process in

ISSN: 2414-3103
DOI: 10.20474/japs-7.1

parallel. But what happens if a user sets the IP address again without waiting for the last process to be finished when there are still unfinished jobs? This can cause race conditions and access violations for the program, which may cause a crash for the binary. It is possible to create a class that handles thread operations in a queue asynchronously to address this issue. When there is a new task to be done, it should be sent to the queue manager class to be executed. If when user wants to leave the page, queue size can be checked. If it is not zero, that means there are still undone jobs, and the user should not leave the page, or all jobs should be canceled, and the user notified accordingly. As it is evident that there may be a timeout for each process, sometimes hardware processes need a restart or to be hung, and this can cause to make a user wait longer in the current menu. The timeout can be added to a queue manager; if time is exceeded, the job can be canceled to overcome this issue.

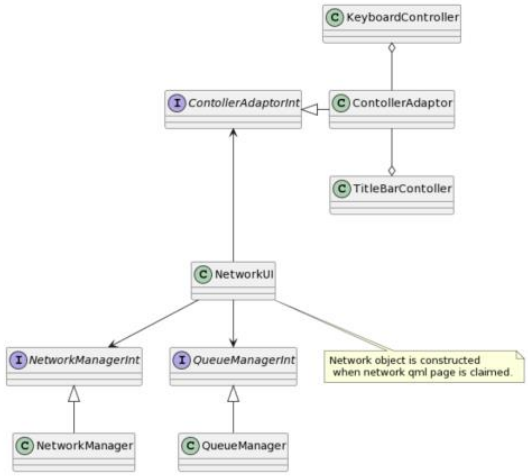

Fig. 11. Class diagram of the Qt application

To handle undesired behaviors during user interaction, QML also can be used. For instance, to set the IP address when the user clicks the "Apply" button in Figure 9 until the hardware is done, it can be deactivated and activated again when the new IP address is set. The same approach also can be used to keep the user on the current page. The back button, which redirects to the main menu, can be disabled, and the user can be notified using a popup screen or title bar. In this way, there can be extra protection for memory or hardware problems that the user can cause.

The design is portable, and the same structure can be applied to other menu items. Unnecessary classes can be removed, and desired ones can be added. For instance, for the User menu item, there can be no need for the Network Manager class, but the file handler class is needed, so using the same pattern, the File Handler class can be added and used efficiently. This will make the whole structure easy to understand and sustainable for maintenance. Additionally, when a user leaves the page, all unused objects will be destroyed, increasing performance from a memory perspective. Unused objects should not be kept; otherwise, it is possible to see performance issues. As a result of the design, the menu item should inherit the classes related only to this item and should have an aggregation relationship with objects already alive for common graphical objects over the interface or a pattern [24] like an adapter [24] pattern. For queue thread handlers, it is possible to use mutex and synchronization primitives. Still, it is preferable to use a lock-free mechanism not to see any effect when the memory model changes from strong to weak or vice versa. For queue thread handlers, it is possible to use mutex and synchronization primitives. Still, it is preferable to use a lockfree mechanism not to see any effect when the memory model changes from strong to weak or vice versa.

ISSN: 2414-3103
DOI: 10.20474/japs-7.1

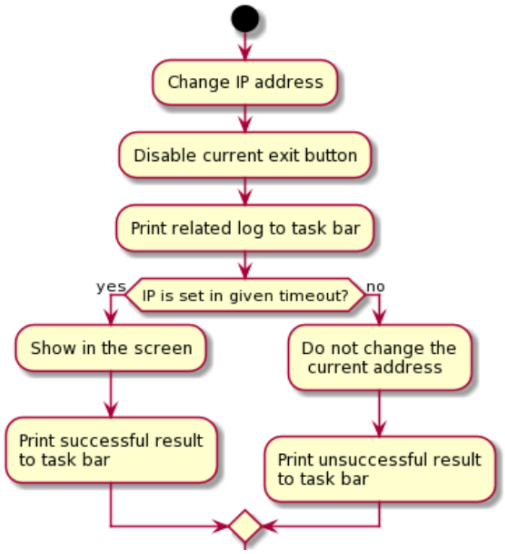

Fig. 12. Example activity diagram

B. Behavior with activity diagram

The simple IP set activity is explained in Figure 10. When a user wants to set or change the device's IP address, it is necessary to open the desired configuration page using the menu items. There can be a simple text field on the page to show and edit the IP address. Set command can be taken from a button, check box, or directly from the text field itself. Many actions should be taken when a set order is requested from the user in the background. First, it is a good idea to disable the exit button for the current page if there is no asynchronous structure. The user should be notified according to the situation so the title bar can be used to do it. From the title Bar Controller class, it is possible to change the text with animation according to the current situation. During this time user can not leave the page. If the user wants to leave the page, then the current process should be destroyed before the Network UI page is killed or destroyed; otherwise, there will be memory issues, which can affect the whole program for the application. This is because, for each page, there is a class object when it is constructed when the page is requested. Simply page lifetime is equal to a class object in a code base. After the given timeout, which is used for the case that when there is a problem during the process, if IP is set, then the title bar should be updated according to the successful result, and the current exit button can be enabled, so the user can safely leave the current page. If, after a timeout, there is no successful result, then the task bar is updated with an unsuccessful result, and the current exit button can be enabled.

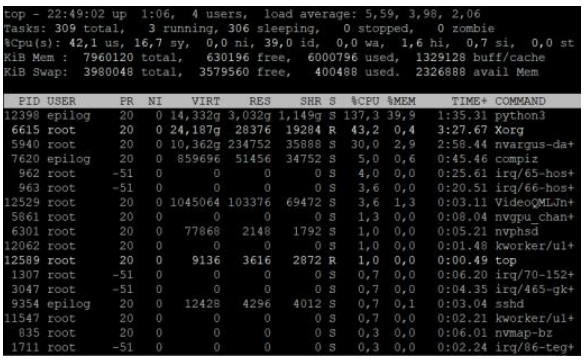

Fig. 13. High CPU usage. Without the design CPU usage for Xorg is really high.

ISSN: 2414-3103
DOI: 10.20474/japs-7.1

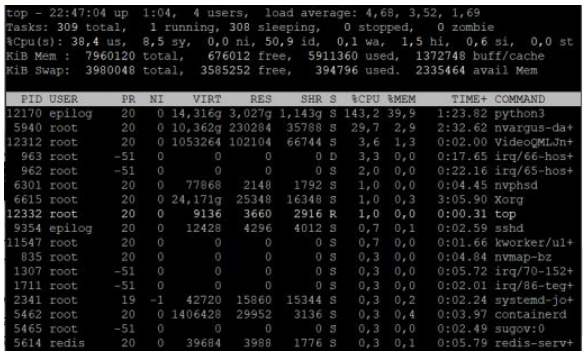

Fig. 14. Low CPU usage. When design is applied, for Xorg, CPU usage is really low.

IV. RESULTS AND DISCUSSION

With this design, it is possible to see in Figure 13 and Figure 14 that there is about a 40 percent CPU usage difference for XOrg. Because of the EGLFS platform, CPU usage can be reduced. In Figure 13, the current design is not used; all windowing representation is handled by the python language. In Figure 14, only AI calculations and algorithms are handled by python, and other windowing processes and other UI based behaviors are handled by the Qt application, which is implemented like the design which is explained in this paper.

Not only CPU usage improvement and also due to the flexibility of QML language, but it is also easy to create industrial user interfaces on the current design with keeping the same efficiency as much as possible during real-time vision [25] data representation on the screen while UI elements are active.

As a consequence of the portable design, the same pattern can be used for user interface development which reduces the number of bugs and development time. It will keep consistency for all UI elements in the application. Furthermore, because of the modular structure, in case of changing the AI model, which can be a different framework or library, it is easy to replace the python AI application with the new one to test or make a demo.

REFERENCES

[1] Github. (2022) Industrial user interface example for Python Visual AI Applications with Qt. [Online]. Available: https://bit.ly/3RmKegM

[2] Wikipedia. (n.d) What is qmls? [Online]. Available: https://en.wikipedia.org/wiki/QML

[3] Wikipedia. (n.d) What is qt? [Online]. Available: https://en.wikipedia.org/wiki/Qt

[4] A. Vaduva, A. Gonzalez, and C. Simmonds, *Linux: Embedded development*. Birmingham, UK: Packt Publishing Ltd, 2016.

[5] G. S. Nagpal, G. Singh, J. Singh, and N. Yadav, ``Facial detection and recognition using opencv on raspberry pi zero,'' in *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. IEEE, 2018, pp. 945-950.

[6] L. Z. Eng, *Hands-On GUI Programming with C++ and Qt5: Build stunning cross-platform applications and widgets with the most powerful GUI framework*. Birmingham, UK: Packt Publishing Ltd, 2018.

[7] M. C. Neto, S. S. Andrade, and R. L. Novais, ``Cross-platform multimedia application development: for mobile, web, embedded and iot with qt/qml,'' pp. 23--26, 2017.

[8] T. Shevgunov and E. Efimov, ``Software implementation of spectral correlation density analyzer with RTL2832U SDR and Qt framework,'' in *Computer Science On-line Conference*. Cham, Switzerland: Springer, 2019.

[9] J. M. Willman, ``Working with Qt Quick,'' in *Beginning PyQt*. Berkeley, CA: Apress, 2022, pp. 359-394.

[10] V. V. Savinykh and O. V. Postylyakov, ``On development of cross-platform software to continue long-term observations with the Brewer Ozone Spectrophotometer,'' in *Remote Sensing of Clouds and the Atmosphere XXIII*. Berlin, Germany: SPIE Remote Sensing, 2018, pp. 212-223.

[11] G. Lazar and R. Penea, *Mastering Qt 5: Create stunning cross-platform applications using C++ with Qt Widgets and QML with Qt Quick*. Birmingham, UK: Packt Publishing Ltd, 2018.

ISSN: 2414-3103
DOI: 10.20474/japs-7.1

[12] N. Sherriff, *Learn Qt 5: Build modern, responsive cross-platform desktop applications with Qt, C++, and QML*. Birmingham, UK: Packt Publishing Ltd, 2018.

[13] R. Shilkrot and D. M. Escrivá, *Mastering OpenCV 4: A comprehensive guide to building computer vision and image processing applications with C++*. Birmingham, UK: Packt Publishing Ltd, 2018.

[14] N. K. Goel, M. Sarma, S. Valluri, D. Agrawal, S. Braich, T. S. Kuswah, Z. Iqbal, S. Chauhan, and R. Karbar, ``Captionai: A real-time multilingual captioning application.'' in *INTERSPEECH: Show & Tell Contribution*, 2019.

[15] S. M. Palakollu, *Practical System Programming with C*. Berkeley, CA: Apress.

[16] L. Z. Eng, *Qt5 C++ GUI Programming Cookbook: Practical recipes for building cross-platform GUI applications, widgets, and animations with Qt 5*. Birmingham, UK: Packt Publishing Ltd, 2019.

[17] Wiki. (n.d) What is xorg. [Online]. Available: https://wiki.archlinux.org/title/xorg

[18] Qt. (n.d) What is eglfs. [Online]. Available: https://doc.qt.io/qt-5/embedded-linux.html

[19] Y. Xiang, Y. Chen, J. Ye, B. Wen, and H. Hu, ``Design of multi-parameter monitoring system based on embedded Linux+ Qt,'' *Zhongguo yi Liao qi xie za zhi= Chinese Journal of Medical Instrumentation*, vol. 44, no. 2, pp. 127-131, 2020.

[20] Qt. (n.d) What is qpa? [Online]. Available: https://doc.qt.io/qt-5/qpa.html

[21] D. Bogosavljev, M. Popovic, and D. Bogdanovic, ``Analysis of a software based hardware composer adaptation,'' in *29th Telecommunications Forum (TELFOR)*. IEEE, 2021, pp. 1-3.

[22] B. Barladian, N. Deryabin, A. Voloboy, V. Galaktionov, and L. Shapiro, ``High speed visualization in the JetOS aviation operating system using hardware acceleration,'' in *CEUR Workshop Proceedings*, 2020.

[23] J. Zhang, X. Tian, and Q. Duan, ``Design and implementation of vehicle terminal graphic interface based on QT,'' *Journal of Physics: Conference Series*, vol. 1486, no. 7, pp. 1-5, 2020.

[24] D. Nesteruk, *Design patterns in modern C++: Reusable approaches for object-oriented software design*. Berkeley, CA: Apress, 2018.

[25] J. Sigut, M. Castro, R. Arnay, and M. Sigut, ``OpenCV basics: A mobile application to support the teaching of computer vision concepts,'' *IEEE Transactions on Education*, vol. 63, no. 4, pp. 328-335, 2020.

ISSN: 2414-3103
DOI: 10.20474/japs-7.1