Interfaces and Software Control Models of Multipoint Aerial Objects Trajectory Measurements System

Andriy Tevyashev
Department of Applied Mathematics
Kharkiv National University
of Radio Electronics
Kharkiv, Ukraine
andrew.teviashev@nure.ua

Anton Koliadin
Department of Applied Mathematics
Kharkiv National University
of Radio Electronics
Kharkiv, Ukraine
anton.koliadin@nure.ua

***Abstract*—This paper is devoted to the development of the software interfaces for multipoint air objects trajectory measurements system.**

***Keywords*—aerial objects tracking, user interface, multi-threading, cross-platform, signals and slots, performance.**

I. Introduction

For optical-electronic system of aerial objects trajectory measurements (OESTM) the development of the interface and software requires the use of the fastest methods to provide sufficiently small delays for the operation of the entire system in real time and the ability to obtain video recording of high-speed supersonic targets.

The programming language is C++, some individual modules are written in C. This provides the highest performance on modern x86-64 compatible processors on Windows and Linux operating systems.

The basis of the user interface is made in the QT framework, which is written in C ++, is characterized by high speed, cross-platform and powerful methods of managing software flows. Each individual computing function that requires real-time performance or is directly involved in data exchange with stations is performed by a separate thread, which independently of other program modules is performed in parallel. Due to the multi-core processors, it is possible to run several threads simultaneously without interruption, each of which loads the entire processor core [1].

Individual threads in QT are implemented on the basis of QThread classes. Each object in QT allows you to transmit signals to the slots of any other object. All elements of the interface are objects implemented on the basis of its class which are the successors of the base class QObject.

II. Signals and slots in QT

Signals and slots are used for communication between objects. The signals and slots mechanism is the main feature of Qt and is probably the main part of Qt that differs the most in functionality from other libraries.

Qt uses an alternative technique to callbacks: signals and slots. A signal is emitted when a certain event occurs. Qt widgets have many predefined signals, and you can always subclass them to add your own signals. A slot is a function that is called in response to a specific signal.

All classes inheriting from QObject or one of its subclasses (for example, QWidget) can contain signals and slots. Signals are emitted when an object changes its state, if this change may be of interest to other objects. All objects do this to communicate with other objects. They don't care if anyone gets the signals they emit. This is a true encapsulation of information, and it ensures that objects can be used as separate pieces of software. Signals are emitted by an object when its internal state changes, and if it might be of interest to its clients or owner. Only classes containing signal definitions and their subclasses can emit signals.

When a signal is emitted, the slots associated with it are executed immediately, just like a normal function call. When this happens, the signals and slots mechanism is completely independent of the GUI event loop. Execution of the code following the emit statement will continue as soon as all slots have finished executing. In the case of queued connections, the situation is somewhat different; the execution of the code following the emit will continue immediately, and the slots will be executed a little later.

The slot is called as soon as the signal connected to it is emitted. Slots are regular C ++ functions and can be called in the usual way; their only feature is that signals can be attached to them.

Since slots are regular member functions, they have the same access rights as regular member functions. However, as slots, they can be called by any component regardless of the access level by means of a signal-slot connection. This means that the signal emitted by an object of an arbitrary class can be associated with a private slot and called in a completely foreign class.

III. QT thread management technologies

Qt provides support for threads in the form of platform-independent threading classes, a thread-safe way to dispatch events, and the ability to establish signal-slot connections between threads. This makes it easier to build portable multithreaded applications and take advantage of multiprocessor machines. Multi-threaded programming is also

a useful paradigm for performing time-consuming actions without freezing the user interface.

The QMutex, QReadWriteLock, QSemaphore, and QWaitCondition classes provide thread synchronization facilities. Although the main idea behind threads is to make threads as parallel as possible, there are times when a thread must stop its current operations and wait for other threads. For example, if two threads simultaneously try to access the same global variable, the result is usually undefined.

The QtConcurrent namespace provides high-level APIs that make it possible to write multithreaded programs without using low-level threading primitives such as mutexes, read-write locks, wait conditions, or semaphores. Programs written with QtConcurrent automatically match the number of threads in use with the available number of processor cores. This means that applications written today will continue to scale when deployed to multi-core systems in the future [2].

IV. Performance Optimization for Real-Time Tasks

The main task of the development is to minimize delays in the transmission of information from some parts of the system to others, that is, from video cameras to servo drives. In view of the high speed of the target, the time for correcting the position of the next camera after receiving the image of the previous one is significantly less than a second, about 200ms.

The second no less important task is to ensure the continuous operation of the image processing modules by carrying out all the interruptions to the input, output and transfer of information to separate streams.

A module executing in a separate thread is responsible for each resource-intensive operation. The main streams are the image acquisition stream for each camera, the image processing stream for each station, the servo control stream, the target trajectory calculation stream, the operator control stream, the telemetry recording stream.

The mechanism of signals and slots QT has certain delays inherent in any interface; therefore it is used at the stage of initialization and system configuration. Directly in the process of recording a trajectory, the main modules use direct access to memory and the functions of accessing variables of the processing flow, the flow of trajectory calculation [4].

V. GUI Interface View

For the convenience of the user, the interface is divided into separate tabs combined into separate groups. The main groups of tabs are settings, telemetry separately for each station, combined video image from all stations.

The number of stations controlled by the program can be adaptively changed in the development environment.

VI. Conclusions

Developed interfaces and introduced approaches can be used for other similar tasks, where speed and ease of use come first.

For the further development of the communication network of OESTM stations, the possibility of connecting several control points is provided, each of which receives a picture and controls a group of stations into a single network.

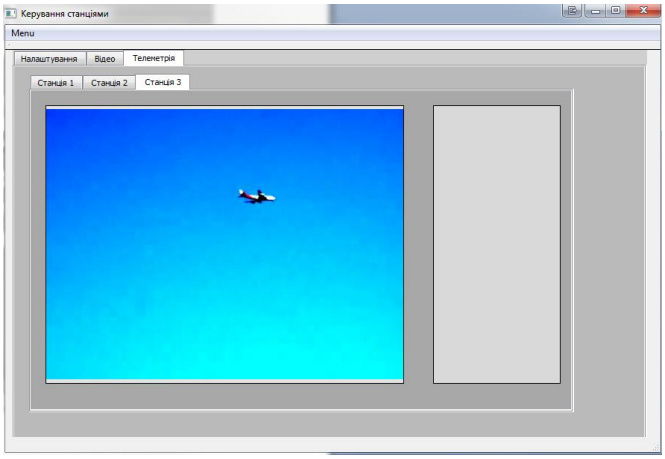

Figure 1. Telemetry picture of an individual station

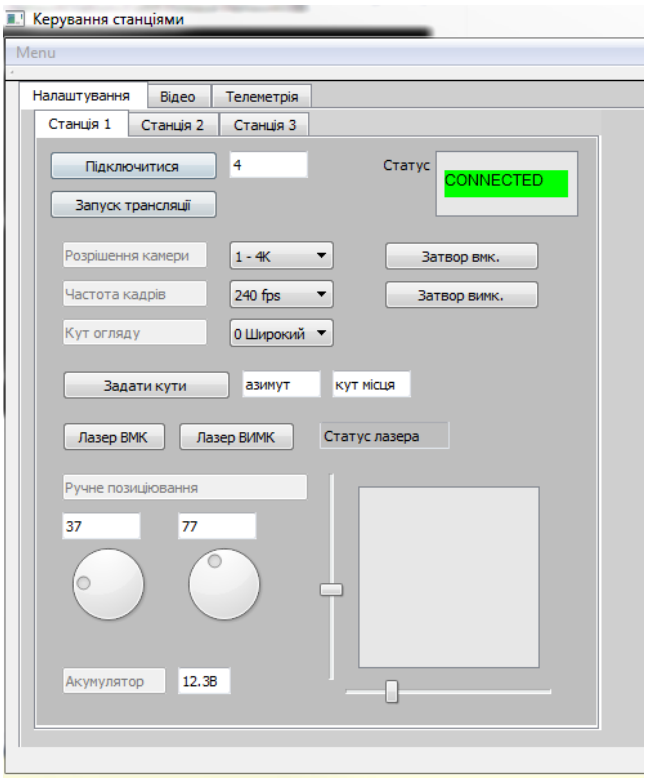

Figure 2. Station settings and control tab

References

[1] Payas Rajan, John Carey, Shreyans Doshi “C++ Data Structures and Algorithm Design Principles” / Packt. 2019. .626p.

[2] R. Murphy, K. Wheeler and D. Thain, "Qthreads: An API for programming with millions of lightweight threads," in *2008 IEEE International Parallel & Distributed Processing Symposium*, Miami, FL, 2008 pp. 1-8.

[3] Dong Y, Han W. (2017) Real-time Research of Linux Operating System Based on Multi-core ARM. Minicomputer System, 38:1262-1266.

[4] Peng J, Shi B. (2010) Construction of Embedded GUI Development Platform Based on Qt. Microcomputer Applications, 26:40-42.