Implementation of object-oriented programming in study of electrical race car

M Nowak$^{1}$ and M Baier$^{2}$
$^{1}$Silesian University of Technology, Faculty of Automatic Control, Electronics and Computer Science, 16 Akademicka Street, 44-100, Gliwice, Poland
$^{2}$Silesian University of Technology, Faculty of Mechanical Engineering, 18A Konarskiego Street, 44-100, Gliwice, Poland

E-mail: maja.baier@gmail.com

Abstract. The paper covers issue of conducting advanced research of electrical race car participating in international competition called Sileverline Corporate Challenge. Process of designing race cars in Silesian Greenpower team is aided by a professional engine test stand built particularly in purpose of this research. Phase of testing and simulation is an important part of the implementation of new technologies. Properly developed solutions and test procedures are able to significantly shorten development time and reduce design costs. Testing process must be controlled by a modular and flexible application, easy to modify and ensuring safety. This paper describes the concept of object-oriented programming in LabVIEW and exemplary architecture of object-oriented control application designed to control engine test stand of the electrical race car. Eventually, the task of application will be to steer the electromagnetic brake and the engine load torque to perform according to data from the actual race track. During the designing process of the car, minimizing energy losses and maximizing powertrain efficiency are the main aspects taken into consideration. One of the crucial issues to accomplish these goals is to maintain optimal performance of the motor by applying effective cooling. The paper covers the research verifying the effectiveness of the cooling system.

1. Introduction
The test stand of the electrical motor is a diagnostic and measuring device designed for examining the drive system and defining various traction properties. The basic types of motor/engine test stands are motor stands which enable accurate testing of the engine parameters directly and chassis test stands which allow indirect measurements without the need of disassembling the motor from the vehicle.

The test stand (figure 1.) described in this paper is unique as far as its modularity is concerned. The design of this device is a combination of motor stand and chassis test stand and allows both types of measurement. The application to control the test stand was written in such way to enable simulation of specific motor circuits’ conditions - going upwards, downwards, with favorable or unfavorable wind direction - thanks to two extra motors performing as a brake and as an amplifier.
2. Description of the test stand

The application is being developed within the Silesian Greenpower interfaculty student research project carried out at Silesian University of Technology. The project team designs and builds electric cars that take part in the annual Greenpower Challenge racing series. Research on the efficiency of the car resulted in building the test stand for complete drive system analysis, including the engine, the drivetrain, the wheel of the car and the ground.

The test stand (figure 2.) consists of:

- DC motor used in the car (main motor)
- Drivetrain (belt)
- Car’s wheel
- Three phase SEW DRE80M4 motor + absolute encoder (supporting motor)
- SEW Eurodrive MDX61B inverter with DFE11B extension card
- Torque sensor
- Ethernet RIO controller

The motor of the car and its wheel are coupled through a belt in the same configuration as in the car itself. The wheel of the car rolls on the rotating shaft. The shaft is loaded with the mass that represents the motor load when the car is driving on the flat track. The shaft and the additional motor are also coupled through a belt. This motor simulates the load variability resulting from the change of the track gradient over time.

With its frame disassembled, the test stand can perform tests on fully assembled car, so that the engine cooling can also be examined. Main task for the application combined with the test stand is to simulate the car driving on the real track, i.e. to provide the engine of the car with the proper load varying over time. The simulation is based on the data gathered from the real tracks.

3. Application - hardware and software platform

The program is implemented in LabVIEW graphical language delivered by National Instruments. All the actuators are controlled by NI Ethernet Rio controller. The controller is connected to local PC.
displaying the front panel of the application that allows to control the test stand’s operation using touch screen.

Figure 2. Main part of the test stand - CAD model and real view.

### 3.1. Application: architecture

The application implements multithreaded architecture with threads separating the tasks within the program. Such an approach is necessary in multitask programs that additionally control actuators like DC motors – the program needs to handle user actions, data logging and visualization of the collected data while not affecting control or measurements tasks. The program consists of threads responsible for the following tasks:

- Error handling
- Collecting the data from FPGA
- Data processing
- Data logging
- Network communications
- Control over actuators

In the presented application, the communication between threads is realized via message queues introducing the producer-consumer programming model, as described in [1]. For instance, each of the threads can produce a message containing the error that occurred during operation and send it to the error processing thread, which operates as a consumer awaiting for messages on its queue. Another example of this model is the control thread consisting of minor loops exchanging commands via message queues. The commands are then being interpreted by the consumer loops introducing another programming model, i.e. state machine presented in [2]. The consumer programs move within predefined operating modes, where the choice of the next state depends on decisions made within the previous state. Termination of thread operation is realized by sending the stop command to all of the threads – after reception of such a command, the loop is stopped.

Respective tasks are divided between separate threads. For example, error handling is centralized within a single error processing thread – all errors are sent on its queue. Such an approach simplifies the code assuring that a specific error is always handled in the same way and that no error is missed. Code replication is thus reduced.

### 3.2. Control and communication with the inverter

The SEW MDX61B inverter operates in field-oriented control mode with torque control using the feedback from the encoder built in the motor. The control is realized by limiting the maximal active
current supplied to the motor within the internal configuration of the inverter. The torque generated by a SEW motor is a linear function of the active current, as claimed in [6]. The torque setpoint is calculated by the RIO controller and applied to the inverter by modifying the active current limit.

To integrate the inverter with the existing test stand, an original LabVIEW library was created. The library contains a set of functions that realize the control over the inverter using Ethernet connection. The library implements the standard way of communication with SEW products consisting of sending three Process Output (PO) words to the inverter and receiving three Process Input (PI) words. The function of those words can be specified within the internal parameters of the inverter. PO words control the inverter operation, PI words can be used for measurement purposes.

The program communicates with the inverter via Ethernet using Modbus/TCP protocol. Communication is provided by the DFE11B extension card of the inverter. The application connects with two different ports of the inverter. One port is used to send and read process words. This is the standard way of communicating with the inverter.

To achieve additional functionality, the library was extended with blocks connecting with the additional port. Those blocks allow the program to read the value of any parameter from the memory of the inverter. Using the additional connection, the application can read the limits of the control signals, actual values of the current applied to the motor or the rotational speed of the shaft.

3.3. Object orientation

The program is object-oriented. Object oriented programming is a model contrary to the procedural programming, where the program can be seen as a set of instructions and subroutines [3]. In object-oriented code, the application design is based on classes, which can be seen as abstract structures describing a problem or a device [4]. The particular realization of the class – an object (also: an instance of the class) – represents the data on which the program operates associated with an existing device. Object-oriented application consists of different objects interacting with each other. In order to function, the object needs its methods – a set of subroutines that can be executed on the object of the specific class [5].

All threads of the program are implemented as objects. They share a set of common mechanisms including error handling and execution termination. Those functionalities are implemented as methods of the parent class and are inherited by all the threads. Thanks to this solution, there is no need to define these parts of code for each of the thread separately.

3.4. Inheritance

In the described program, there is a class representing a generic thread. Its model assumes the basic variables describing the thread, like thread name or execution period. It implements the fundamental methods responsible for thread initialization, error handling and reading configuration from a file. There are also more specific classes – like error handling thread class or data processing thread class. The basic idea of class inheritance is that the inheriting class includes (inherits) all the data and methods of the parent plus extensions applied to that specific class, as presented in [4]. Thus, the code common for all objects (the parent class and inheriting classes) does not have to be replicated for all of the inheriting classes. When a specific method is called, LabVIEW checks if the current class implements this method. If it does, the method of the class is executed. If it does not, LabVIEW checks all the direct parents in the inheritance branch recursively and executes the method of the first parent that implements it. If there are many devices that need to be initialized in the same manner, startup method is implemented in parent class and then inherited by more specific classes. There is no need to define this method for all of the devices separately.

On the top level of the inheritance diagram there is always the most general class [4]. It implements methods used by all the inheriting classes. With each level of the diagram, the classes become more specific.
3.5. Advantages of object-orientation

Main advantage of object-oriented programming apart from time efficiency improvement and better code organization is that the program can switch to work with new device model without major changes made to the code [3]. MDX61B inverter is represented as class within the program. The flow diagram is defined for a general object (abstract inverter), and not for each particular device separately. Control over a specific inverter is provided through methods depending on the object passed to the input of the control loop. In the procedural programming, a separate control loop would have to be created for each device and the program would switch between them. This increases the time needed to develop the program and forces to replicate the code. LabVIEW makes it possible to define an array of objects and control several devices from within a for loop, with only slight changes made to the program code.

Object-oriented approach also makes the program less dependent on the device model used on the test stand and makes it possible to exchange the device for another model with only a little work needed for the application to resume operation. Adding another devices or modifying the code shared by similar devices is far less time consuming.

3.6. Safety measures

A command stopping the motors is always sent just after the application startup. Controls on the front panel are locked, excluding the button stopping the drive that is always unlocked. To unlock the controls and start the motor, operator has to confirm, that the test stand is safe. The purpose of such a solution is to remind the user to ensure that no one is working with the test stand at the moment (test stand is equipped with a webcam monitoring the mechanical equipment). Only when user confirms that the test stand is safe to run, the controls on the front panel are unlocked.

Every time the program is started, the user shuts down the program or any error occurs, the inverter is always stopped. The user can stop the motor manually at any time using the program or the mechanical safety button.

4. Thermal research

One of the types of tests carried out on the motor test stand is thermal analysis. The fundamental thermal measurement of the motor parameters was conducted in order to verify the effectiveness of present cooling system. Firstly, the verification of the passive heat sink made of copper and mounted on the motor housing was carried out. The study was to compare the parameters on the test stand in two different conditions: without any cooling and with copper heat sink and to compare results. There was no additional load, only the mass of the test stand shaft. During the test the temperatures were measured via three methods: thermocouples, digital thermometers and infrared camera. Furthermore, current, voltage drop and speed of rotating wheel were measured [7].

Infrared camera images (figure 3.) were taken from the front (shaft) and back (brushes) of the motor, digital thermometers were mounted on two motor covers, thermocouples were mounted inside the motor. Quite an important aspect was the placement of the thermocouples so that they would gather information from the most inner part of the motor and in the same time it would not be any rotating part and the motor could still be closed - to reduce any significant interference in the motor design. Finally, they were attached to the brass tube in which motor brushes are placed. These elements have direct contact with the hot brushes which occurred to be one of the most troublesome heat generators.

The results of thermal research were in the form of graphs of temperature on the brushes and motor covers, graphs of current and in the form of images from infrared camera. Pictures above show the photos taken after 60 minutes of no load performance. Thanks to the heat sink the temperature on the housing near the shaft decreased by more than 11°C and on the opposite side of the housing by more than 14°C. Analysis of the temperature graphs collected by thermocouples showed a 14.5% improvement in maximum values of temperatures of brass parts in the motor with heat sink comparing
to the motor without cooling. What is more, the speed of the wheels increased by 4.5 km/h after the application of cooling.

![Figure 3](image)

**Figure 3.** Pictures from infrared camera after 60 minutes of no load performance a) test without a cooling system b) test with a cooling system.

5. **Conclusion**

The article described the test stand for testing the drive system of an electric car, the object-oriented programming model and its advantages in research applications. Proposed architecture allows to build a reliable and versatile applications capable of carrying out various tasks. Different program solutions introduced to provide safety were also described. The particular problem covered in the paper was thermal efficiency of the motor. The test stand occurred to be a suitable tool to verify computer analysis of the motor and also to conduct a research examining different heat sinks.

**References**

[1] National Instruments Application Design Patterns: Producer/Consumer: http://www.ni.com/white-paper/3023/en/ (accessed 03.2016)

[2] National Instruments Tutorial: State machines: http://www.ni.com/tutorial/7595/en/ (accessed 03.2016)

[3] National Instruments LabVIEW Object-Oriented Programming FAQ http://www.ni.com/white-paper/3573/en/ (accessed 03.2016)

[4] National Instruments manual: Creating LabVIEW Classess http://zone.ni.com/reference/en-XX/help/371361J-01/lvconcepts/creating_classes/ (accesses 03.2016)

[5] Bitter R., Mohiuddin T., Nawrocki M.: LabView: Advanced Programming Techniques, Second Edition, CRC Press 2006, pp. 449

[6] SEW MDX60B/61B System Manual: http://download.sew-eurodrive.com/download/pdf/16838017.pdf, p, 318 (accessed 03.2016)
[7] M. Baier, M. Dziewior, J. Franiasz, M. Zuk, "Thermal Analysis of a DC Motor Cooling System Availing an Engine Test Stand of Electrical Race Car", Applied Mechanics and Materials, Vols. 809-810, pp. 1145-1150, 2015