

Rapid control prototyping of a Hovercraft

Silvano Balemi*, Roberto Bucher*, Paola Guggiari*, Ivan Furlan*, Markus Kottmann[°], Jacques Chapuis[°]

*) Department of Computer Science and Electronics, University of Applied Sciences of Southern Switzerland, 6928 Lugano-Manno, Switzerland ({silvano.balemi, roberto.bucher, paola.guggiari, ivan.furlan}@die.supsi.ch)

[°]) Institut für Mess- und Regeltechnik, Swiss Federal Institute of Technology (ETH), 8092 Zurich, Switzerland (kottmann@imrt.mavt.ethz.ch, chapuis@imrt.mavt.ethz.ch)

Rapid prototyping control systems play a central role in the development process of mechatronic systems. Rapid controller prototyping can be reduced to the software aspects by appropriate choice of control hardware. In this paper we show how a commercial PLC tied to Matlab/Simulink has been used to guide a hovercraft.

1 INTRODUCTION

In mechatronic systems, control issues play a central role. Verification of system choices must be performed as soon as possible in the development process. It is therefore often necessary to be able to check the desired performance at an early stage through the extensive use of rapid prototyping control systems.

In the reported project the control system of a high-performance hovercraft prototype built by Officina Borra SA had to be developed.



Figure 1. Hovercraft on the lake of Lugano

1.1 Control of hovercrafts

Hovercrafts are underactuated amphibious vehicles. Usually, a horizontal fan creates an air cushion sustaining the vehicle, while a propeller produces the drag and a large rudder is used to guide the vehicle by the deviation of the air flow.

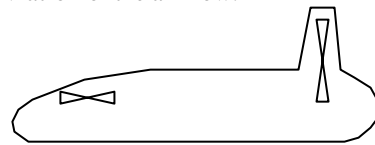


Figure 2. Hovercraft with sustainment fan and propeller

Curved trajectories are followed by letting the vehicle drift laterally: then the direction of the speed does not coincide with the axis of the vehicle.

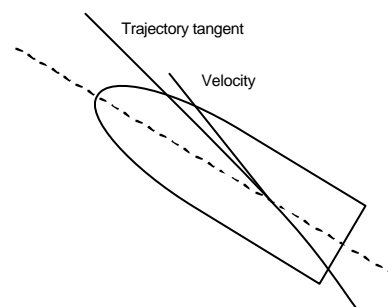


Figure 3. Hovercraft with drift angle with respect to trajectory

Besides the “classical” actuators (rudder, fan and propeller), our vehicle has also 2 air jets on each side and 4 controlled air outlets on the bottom.

The side jets may be used for trajectory control, yaw angle stabilization or for lateral motion during bank approach, while the air outlets on the bottom may be used for actively damping the oscillations related to the pitch and roll angles.

2 THE CONTROL STRATEGIES

There are several possible strategies for controlling the motion of a hovercraft. Assuming to use also the side jets for trajectory control or stabilization, a sample list of the possibilities are:

- control the drift angle with the rudder, use side jets to stabilize oscillations of the yaw angle,
- control the drift angle with the rudder and the front jets,
- create a centripetal force with the rudder and with a front jet (until saturation of the front jet), then control the drift angle.

Such control strategies require the availability of several data, like drift angle, lateral acceleration, forward and lateral speed.

3 THE CONTROL SYSTEM

Because the behavior of a hovercraft is not intuitive and because many actuators are present (the speed of the propeller and the blade pitch should also be dynamically set) it was considered necessary to control the hovercraft in closed-loop.

A joystick is used to set the forward speed and the lateral acceleration. The drift angle and other measurement data are provided by a sensor box developed at ETH Zurich by M. Kottmann and J. Chapuis. The sensor box is an integrated navigation solution containing a GPS sensor, a magnetometer, and an inertial measurement unit (IMU) consisting of 3 accelerometers and 3 gyroscopes. Electrical motors control the throttle of the side air jets and of the air outlets on the bottom. Other electrical motors control the propeller blade pitch angle and the engine throttle.

Instead of cabling each actuator and the sensor box directly to the processing unit, we decided to connect all units together with the help of a field bus. After analysis of different alternatives (existent buses, ad-hoc bus) the CAN bus was selected.

A fundamental choice was the use of commercial hardware wherever possible. The control processing

unit chosen, a programmable logical controller (PLC) from Selectron, presents two CAN bus interfaces and enough computing power for the control needs (a sampling frequency of 25 Hz was deemed to be sufficient).

Linear motors from LinMot were selected. Linear motors, even if not the best choice in a commercial product, allow to simplify the mechanical construction of the air throttles and are a reasonable choice when considering the flexibility needed to perform changes in the mechanical design of a prototype. Finally, the sensor box was also provided with a CAN bus interface.

The complete scheme for the control system is shown in Figure 4.

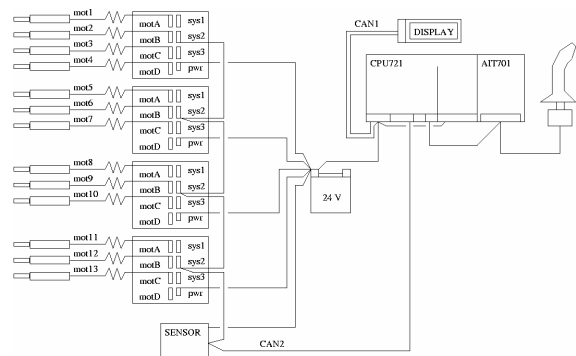


Figure 4. Scheme for the control system with actuators, sensor box, PLC and interfaces.

4 THE CONTROL PROGRAM

Because of the many strategies to be tested, several alternative controllers had to be implemented. In order to decrease the risk of programming errors possibly leading to accidents, automatic code generation was deemed necessary.

4.1 Code generation from Matlab

The most widely used tool for the simulation of control systems is Matlab/Simulink by Mathworks Inc. It was an obvious choice to try to generate the code for the PLC directly from the controller tested in Simulink.

The extension of Matlab/Simulink with the toolbox RealTimeWorkshop (RTW) makes it possible to

automatically create generic C-code from a Simulink model. Within the reported project we adapted the generation of the C-code to match the needs of the PLC, making it possible to run a controller previously tested with a simulation just in a few mouse clicks.

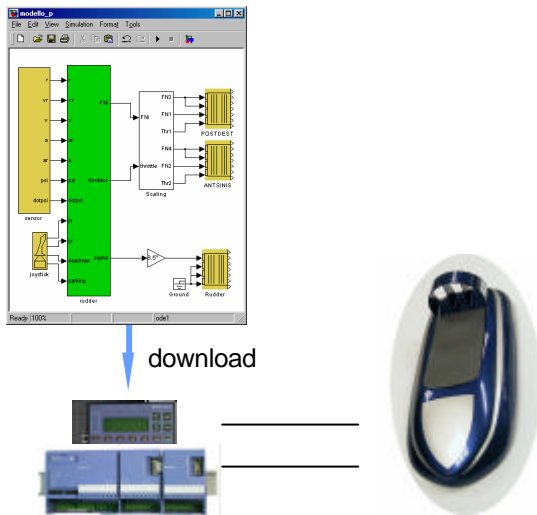


Figure 5. Automatic code generation for the PLC controller from Matlab/Simulink

4.2 The link between Simulink and the PLC

Usually, the C-Code generation through the Matlab Toolbox “RealTimeWorkshop” requires two specific files:

- The “Target Language Compiler” File
- The “Template Makefile”

The role of the “Target Language Compiler” file is to define the environment where the Code is integrated (program language, environment variables, control of the code generation). The “Template Makefile” describes all the information needed to compile the code (path, compilation flags) and is the basis for creating the “Project Makefile”.

In our case, the integrated development environment (IDE) for the Selectron PLC dynamically generates an own Project Makefile each time the compilation is started; thus we can't exploit the external Makefile created by Matlab/Simulink. Moreover, it is not possible to add external files to the project, because the IDE is not able to recognize and to correctly link them. Even the command “gmake -f model.mk” does not work correctly.

Thus, the standard procedures offered by Real-TimeWorkshop were not completely applicable and we had to look for new ways to complete and integrate the generated code.

In spite of all these limitations we succeeded, although with some loss of flexibility, in generating code for the PLC environment, and we completely automated the code generation from the Simulink model to the code downloaded into the Selectron PLC.

4.2.1 Solutions to circumvent the IDE-specific limitations

In order to solve these problems we had to:

- use a preselected name for the code generated by RTW (modello.c), and to integrate it with the line “#include modello.c”,
- include the code for the interface between the PLC and the Simulink in a file (main_166.c) and to integrate it with the line “#include main_166.c”,
- include all source files from RTW in the same way,
- create the drivers to handle the I/O as so-called TLC files in order to avoid external sources (consequently it is impossible to write C-MEX S-Function to handle I/O functions),
- integrate definition for the files from the “State Flow” toolbox using “#include” statements

In order to correctly compile the Simulink files we further had to modify the file defining the “Makefile”. This file, called “Ips.mk” contains all flags and the path needed to compile and link the project. In particular we had to add some new definitions for the Matlab and the RTW environment. The file is given below:

```

MAT_DIR=c:\matlabr12
MAT_INC=-I$(MAT_DIR)\simulink\include -
I$(MAT_DIR)\extern\include
RTW_INC=-I$(MAT_DIR)\rtw\c\src -
I$(MAT_DIR)\rtw\c\libsrc

MATLAB_D=-DMODEL=MODELLO -DRT -DDSP32 -
DNUMST=1 -DNCSTATES=0 -DTID01EQ=0
...
ifreq '$(BOARD_TYP)' 'CPU721'
BINC     =$(BASE)/inc $(MATLAB_I)
...
CDEFS=-D__HWTYP_CPC721__ $(MATLAB_D)

```

4.2.2 The steps for the code generation

It was now possible to simply generate the PLC code according to the following steps:

- Design the control system with Simulink and Stateflow,
- Generate the C-Code using RTW (modello.c, modello.h, modello.reg and modello.prm), see figure 6,
- Compile the code in the Selectron IDE, see figure 7,
- Download the code to the PLC from the Selectron IDE.

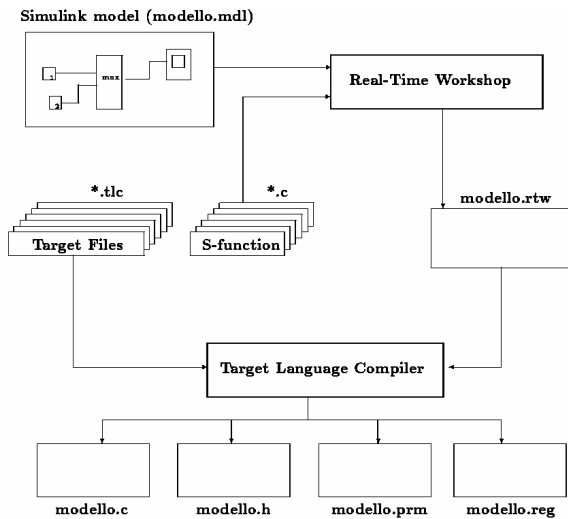


Figure 6. Automatic code generation for the PLC controller from Matlab/Simulink

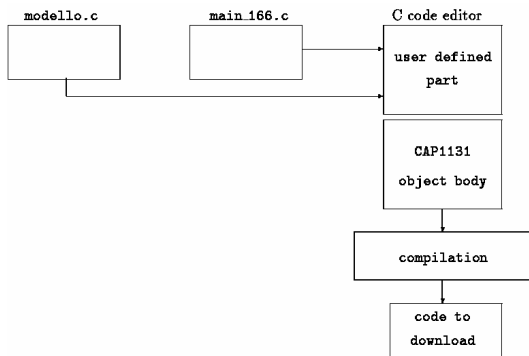


Figure 7. Selectron compilation process

4.2.3 I/O blocks

All PLC I/O blocks are implemented in Simulink as C-MEX S-Function and TLC modules (see Figure 8).

The C-MEX S-Function is only responsible for defining the different inputs and outputs. The C-Code to implement the function of a single module is defined in corresponding “TLC” files.

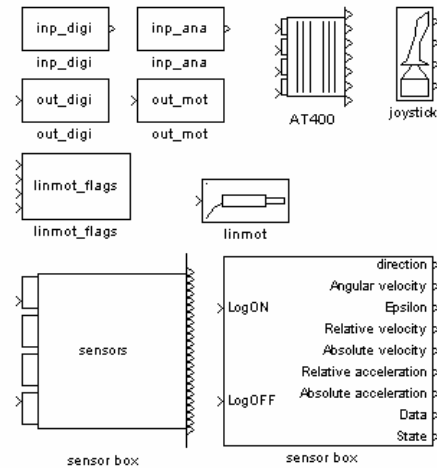


Figure 8. Simulink library for I/O blocks

4.3 Integration with additional code

The PCL C-code is divided into two parts:

1. The external code generated by Simulink
2. The C-code written directly in the Selectron IDE

The code generated by Simulink is responsible for:

- computing the motor positions from the sensor data and pilot inputs according to the desired control policies,
- monitoring the status of the motor drives and of the sensor box.

The C-code generated in the Selectron IDE handles the

- PLC display,
- the link between analog and digital PLC inputs and outputs,
- the communication over the CAN Bus.

In order to exchange data between the Simulink and the PLC code global variables are used.

5 CONCLUSIONS

The reported work shows how rapid control prototyping can be used to speed up the development process of control systems and to increase reliability during the testing phase. Current software tools allow the use of these techniques on the definitive hardware, simplifying the transition from the prototype to the future product.

The implemented system has been applied to and tested on a hovercraft. Extensive testing of the controlled behavior of the vehicle was not possible because a new mechanical design was soon recognized to be necessary. However, the presented control system will certainly speed up the testing phase of the hovercraft and build the core for the control hardware present in the future product.

REFERENCES

1. R. Hayashi, K. Osuka, T. Ono, Trajectory control of an air cushion vehicle, Proceedings of the IEEE/RJS/GI International Conference on Intelligent Systems, Vol.3, 1994
2. C. Eck, J. Chapuis, H. P. Geering, "Software-Supported Design and Evaluation of low-cost Navigation Units," Proceedings of the 8th Saint Peterburg International Conference on Integrated Navigations Systems, pp. 163-172, St. Petersburg, Russia, May 2001.
3. C. Eck, H. P. Geering, S. C. Bose, "Model Based INS/GPS Navigation," Proceedings of the 7th Saint Peterburg International Conference on Integrated Navigations Systems, pp. 95-102, St. Petersburg, Russia, May 2000.