Riid, A., Preden, J., Pahtma, R., Serg, R., and Lints, T. (2009). Automatic code generation for embedded systems from high-level models. *Electronics and Electrical Engineering*, 95(7):33–36.

A. Riid, J. Preden, R. Pahtma, R. Serg, and T. Lints, "Automatic code generation for embedded systems from high-level models," *Electronics and Electrical Engineering*, vol. 95, no. 7, pp. 33–36, 2009.

```
@article{RiidEA09_CodeGen,
  author = {A. Riid and J. Preden and R. Pahtma and R. Serg
and T. Lints},
  title = {Automatic Code Generation for Embedded Systems from
High-Level Models},
  year = {2009},
  journal = {Electronics and Electrical Engineering},
  volume = {95},
  number = {7},
  pages = {33-36}
}
```

ELECTRONICS AND ELECTRICAL ENGINEERING

ISSN 1392 – 1215 *2009. No. 7(95)*

ELEKTRONIKA IR ELEKTROTECHNIKA

T170 *ELECTRONICS*

ELEKTRONIKA

Automatic Code Generation for Embedded Systems from High-Level Models

A. Riid, J. Preden, R. Pahtma, R. Serg, T. Lints

Research Laboratory for Proactive Technologies, Department of Computer Control, Tallinn University of Technology, Ehitajate tee 5, Tallinn 19086, Estonia, phone: +3726202108; e-mail: andri.riid@dcc.ttu.ee

Introduction

The article describes a case study of automatic code generation for embedded systems from high-level models using the Gene-Auto code generator developed within the scope of the ITEA Gene-Auto project (ITEA05018).

Real-time embedded systems have been gaining an increasingly important place both in industrial systems and also in other software-intensive systems. These systems are primarily dependent on microprocessors and software in fulfilling the tasks set upon them. Different constraints are set upon such systems depending on the application domain while in most systems the computational power of the system is limited, putting serious constraints on the feasibility of certain control strategies. It is the case in very many modern systems that the disciplines of system and software engineering must improve to deal with increasing complexity of system functionality as requirements for higher integration of functions, increasing performance, higher levels of autonomy and high level of dependability (reliability, availability and safety) must be met.

It should be noted that the real-time embedded systems have been becoming more and more complex over time. For automotive systems this is due to several factors: safety and reliability requirements, increasing legislative constraints on pollution and on-board diagnostics, etc. For most systems, designs for new systems are constrained by expectations for reducing cost and time to market. Such complexity gradually enters the systems development phase, with ever increasing requirements and translates directly into a growing number of software functions.

The objective of the Gene-Auto project was to develop an efficient open source prototype for certified code generator that is able to generate code from Matlab Simulink and Stateflow models as no certified code generator for this task is available at the moment. As indicated in [3] an automatic code generation could introduce major productivity improvements in automotive software generation by the ability to have industrial software code available on the target processor with a reduced amount of development time. A certified code generator can shorten the development cycle in areas where the software must also carry a certification. A certified code generator allows generating validated code from models validated using desktop based established design environments, which means that there is no requirement to check the integrity and correctness of the generated software code. The transformation from model to code is proven to be correct by means of modern formal verification techniques which are in line with developed strategies [2]. Any change in system requirements can be first integrated to the high-level model where also the desired effect can be verified, after which code can be generated from the model and integrated to the target platform. Code generated from a high-level model has several advantages over hand-written code, for example code maintainability is improved - when enhancements or defect fixes are made, the code is regenerated (and re-optimized) from scratch [3]. Also code reuse is improved as the lack of platform detail embedded in a design means that cross-platform reuse of designs and tests are enabled [3].

The generated code must have the characteristics of professionally written hand code [4]. Naturally the generated code integration to legacy code should be seamless as not all the code will be auto generated. The task of the team at the Laboratory for Proactive Technologies at Tallinn University of Technology in the Gene-Auto project was to develop test cases to be used for verification and validation of the code generator. Two test cases were developed during the project by our team, one of which will be described in detail in the current article.

Workflow

The workflow for using the Gene-Auto code generator does not change the workflow of modern software development cycle much. A high-level model of the system or a component of the system is developed in a high-level modeling environment such as Matlab. After the model has been verified in the modeling environment and deemed suitable for deployment on an embedded system, code is generated to match the high-level model. The steps for the model evaluation may differ in various industries – in some industries simulation is used as part of the model verification and validation process while other steps are also possible. Currently the code generation step used by the industry in general is manual as the code generation tools available are not certified and for this reason the use of such tools is not feasible in areas where the software must carry industry certification. The Gene-Auto code generator removes that manual step as the code generator itself can be certified using industry standard certification procedures which means that once a high-level model has passed the verification step, verified code can be generated from that model using the Gene-Auto code generator.

Description of the test-case

The test case described in the current paper is a four-wheeled mobile robot that is capable of positioning itself in an indoor environment using an indoor positioning system. The robot is equipped with several embedded computers and the task of the robot is to move from an arbitrary start position to a desired end position while avoiding obstacles.

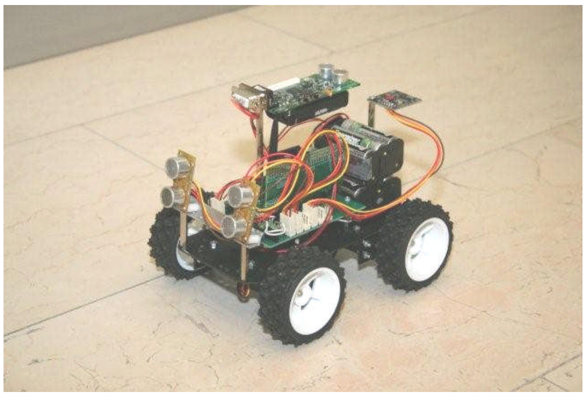

Fig. 1. Mobile platform

In front the mobile platform is equipped with 2 SRF08 ultrasonic rangers for detecting obstacles in the range of 3 – 600cm. In the center of the robot there is an electronic (MEMS) compass CMP2X from Mindsensors for determining the orientation of the robot. A Cricket MCS41CA positioning system receiver from Crossbow enables positioning of the robot. The complexity of sensor interfaces and servomotor control is contained in the Hardware Abstraction Layer (HAL), which allows for generalized control algorithms to be applied on the mobile platform with minimum or no modifications. HAL consists of Atmel AVR based Robostix board from Gumstix and corresponding firmware that interfaces with actuators and sensors. The Robostix board provides PWM signal to servos and I2C bus master service for ultrasonic rangers and compass. The HAL allows for 4 speeds forward and 4 speeds backwards both for left and right side motors. This configuration gives the mobile platform flexible maneuvering abilities of the robot. The HAL is interfaced to higher level hardware via a serial interface.

The higher-level control algorithms run on a Gumstix Connex 400 board based on a 32bit Intel XScale processor running custom Linux OS. This board is interfaced to HAL and the positioning system node by serial interfaces. The software code generated from high-level models is loaded to this sub-system.

Description of the test-case Matlab model

Robot position is determined by three state variables x, y and Φ = [-90°, 270°], where the latter is the angle between robot's onward direction and the x-axis (Fig. 2) in an established coordinate system. The width and length of the robot are denoted by w and l, respectively.

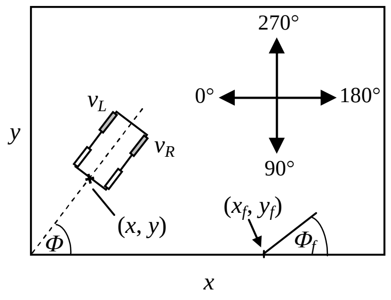

Fig. 2. Robot and its main variables

The problem is formulated as follows: the robot must arrive from an arbitrary initial position (x_i, y_i, Φ_i) to the predefined destination (x_f, y_f, Φ_f). The robot moves forward with the speed that is determined by speeds (tangential velocities) of front wheels, v_L and v_R. These two parameters also provide the means for controlling the robot and our task is to define appropriate profiles of v_L and v_R throughout the control cycle.

For robot movement simulation, the following set of equations (1) is implemented in MATLAB.

$$\begin{cases} \Phi_{t+1} = \Phi_t + \dfrac{(v_R - v_L)}{w}\Delta t, \\ x_{t+1} = x_t + \dfrac{(v_R + v_L)}{2}\Delta t \cos\Phi_{t+1}, \\ y_{t+1} = y_t + \dfrac{(v_R + v_L)}{2}\Delta t \sin\Phi_{t+1}. \end{cases} \tag{1}$$

Implementation of the control task is based on the ideas of [5], and is distributed between two units (Fig. 2). The main component of the system is the fuzzy logic based trajectory mapping unit (TMU) that specifies the optimal robot angle (Φ_r) for the given point in input space determined by its current coordinates x and y. Computation of speeds v_L and v_R that would force the robot to take desired orientation is carried out by a separate steering block SB.

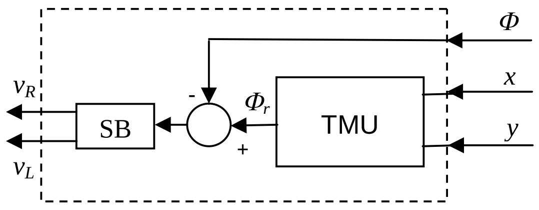

Fig. 3. Hierarchical control system

While the TMU used in [5] can be used without modification for current application, the steering block in [5], however, originally consisted only of a PD controller because in this application the robot was controlled by a steering angle θ of front wheels. For differential steering control scheme in current application this is complemented by a calibration block that computes appropriate v_R and v_L based on θ from the PD controller output using the following equations

$$\begin{cases} v_R = \dfrac{v_{\max}}{2} + \dfrac{v_{\max} \cdot \theta}{2 \cdot \theta_{\max}}, \\ v_L = \dfrac{v_{\max}}{2} - \dfrac{v_{\max} \cdot \theta}{2 \cdot \theta_{\max}}. \end{cases} \tag{2}$$

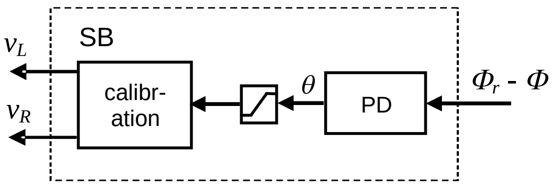

Fig. 4. Contents of the steering block

Note that the maximum forward speed of the robot is determined by $v_{\max}/2$.

The resulting control module can be used for robot control in backward driving mode in its current shape and in forward driving mode if we correct Φ_r by 180° (the reason is obvious) and multiply the SB outputs by –1. Its performance was validated by simulations in MATLAB.

Code generation

The Gene-Auto code generator uses the Gene-Auto modeling language to which all input models are converted. The purpose of this language is to generalize the common concepts found in the tools used for developing safety critical embedded systems and provide them with clean and rigorous semantics. The definition of the intermediate language includes abstract syntax and semantics, but also modeling and programming rules which should ensure that the models are well formed [1]. From the model representation in the intermediate language code is generated for the target language, currently code can be generated in C language. Currently two modeling environments can be used to generate input models for the Gene-Auto code generator – Matlab Simulink/Stateflow and Scicos. In the case study the code was generated from the controller part of the Matlab model. The code generation is straightforward – the code generator is given the Matlab model and code is generated from the model. When a change in the model is made (whether because of change of requirements or some other reason) the code can be generated again and integrated to the existing code.

Generated code integration

Gene-Auto allows for very flexible interfaces of the generated code. Interaction between platform specific code and generated code can be performed either via global variables defined as extern in generated code or via data structures passed to functions in the generated code. The generated code can be contained in one source file or distributed into multiple files, based on the structure of the source model. The structure of the generated code does not influence the integration of the code, because the entry point is in one file. The required header files are included automatically for convenience. As a result, the higher-level code to which the generated code is integrated only needs to include one generated header file where the interface for the generated code is defined. The generated and manually written code must be compiled and linked. For our test case GNU Make was used for automating the build process, allowing an easy choice between different versions of generated code, target hardware and component configuration. There is obviously no need to change the top level code when the generated code changes if the interface remains constant and different versions of the generated code are stored in different folders. Make options can then be used to choose which version is compiled and linked into the final executable or library. The compiled code was run at the mobile platform and the behaviour of the platform corresponded to the behaviour of the Matlab model.

Code comparison between generated code and manually written code

The generated code was compared to manually written code with equivalent functionality. Various parameters of the generated code and manually written code were compared to determine the differences.

As the first step the code size was evaluated. As it can be seen the code size of the generated code was greater which is not that important given the fact that ROM is generally cheap.

Table 1. Code size of "the control of a mobile robot" case study

	Number of files	Number of statements	Number of lines of code	Number of functions
handwritten	28	789	1549	27
generated	10	1539	2320	5

Since the compiled code runs under the Linux operating system the memory usage values were monitored under the Linux operating system using the operating system's ps utility. Both memory usage values are for a process that runs the test program used to evaluate the generated code and the hand-written code. The test program contains some wrappers that call the actual code and evaluate the returned results. Since the test program was identical in both cases (except for the code under evaluation), any change in memory usage is a result of the application code change.

Table 2. Memory usage of "the control of a mobile robot" case study, in kB

	Non-swapped physical memory	**Virtual memory size**
handwritten	1180	16 040
generated	1252	16 048

The CPU performance was analyzed by measuring the execution time of the code over a 1000 executions of the code. The values in the tables are execution time of the code in nanoseconds.

Table 3. Execution times of "the control of a mobile robot" case study, time in nanoseconds

	Minimum	**Average**	**Maximum**
handwritten	2 933	3 517	26 959
generated	8 590	10 848	81 016

The traceability between model and code was checked by reading the code and it was found to be satisfactory. Since the structure of the code does not follow the structure of the model the traceability is hindered. However the comments in the generated code provide a sufficient level of traceability between the model and the code.

Concluding remarks

Code generation for safety-critical real-time embedded systems is viable and important if we want to cope with the complexities of modern systems. The case study showed that while the generated code may be slightly bigger it satisfies the requirements for numerical accuracy and functionality.

Acknowledgements

The work presented in this paper was partially supported by the ITEA project 05018 Gene-Auto, the Estonian Science Foundation grants ETF6182, ETF6837 and the Estonian Ministry of Education research grant SF0140113As08.

References

1. **Toom A., Naks T., Pantel M., Gandriau M., Wati I.** GeneAuto: An Automatic Code Generator for a safe subset of SimuLink/StateFlow // Dans: European Congress on Embedded Real-Time Software (ERTS 2008). – Toulouse, 29-Jan-08–01-Feb-08. – Société des Ingénieurs de l'Automobile (support électronique). – 2008.
2. **Rugina A. E., Thomas D., Olive X., Veran G.** Gene-Auto: Automatic Software Code Generation for Real-Time Embedded Systems // Data Systems In Aerospace DASIA2008. – Spain 2008.
3. **Weigert T., Weil F., van den Berg A., Dietz P., Marth K.** Automated Code Generation for Industrial-Strength Systems // Computer Software and Applications COMPSAC '08. 32nd Annual IEEE International Conference. – July 28, 2008 – Aug. 1, 2008. – P. 464–472.
4. **Toeppe S., Bostic D., Ranville S., Rzemien K.** Automatic code generation requirements for production automotive powertrain applications // Computer Aided Control System Design. Proceedings of the IEEE International Symposium. – 22–27 Aug., 1999. – P. 200–206.
5. **Riid A., Pahhomov D. and Rüstern E.** Car Navigation and Collision Avoidance System with Fuzzy Logic // Proc. IEEE International Conference on Fuzzy Systems. – Budapest, 2004. – Vol. 3. – P. 1443–1448.

Received 2009 04 27

A. Riid, J. Preden, R. Pahtma, R. Serg, T. Lints. Automatic Code Generation for Embedded Systems from High-Level Models // Electronics and Electrical Engineering. – Kaunas: Technologija, 2009. – No. 7(95). – P. 33–36.

Software engineering methods have not been able to keep up with the complexities of modern real-time embedded systems. Automatic code generation from high-level formal models removes the manual coding step, enabling faster and higher quality code generation from system specification. A code generation case study involving a mobile robot control algorithm is described and verified in MATLAB, performance and properties of the generated code are compared to hand-written code. Ill. 4, bibl. 5 (in English, summaries in English, Russian and Lithuanian).

А. Рийд, Ю. Преден, Р. Пахтма, Р. Серг, Т. Линтс. Автоматическое генерирование кода для встроенных систем с использованием модели высокого уровня // Электроника и электротехника. – Каунас: Технология, 2009. – № 7(95). – С. 33–36.

Описывается способ автоматического генерирования программного кода для обработки данных в реальном времени. Предлагается алгоритм управления роботом с использованием MATLAB. Дано сравнение сгенерированного кода с кодом, созданным программистом. Ил. 4, библ. 5 (на английском языке; рефераты на английском, русском и литовском яз.).

A. Riid, J. Preden, R. Pahtma, R. Serg, T. Lints. Automatinis įterptinėms sistemoms skirto kodo generavimas naudojant aukštojo lygio modelius // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2009. – Nr. 7(95). – P. 33–36.

Klasikiniai programinės įrangos kūrimo metodai netinka modernioms sudėtingoms duomenis realiu laiku apdorojančioms įterptinėms sistemoms kurti. Automatinis programinio kodo generavimas naudojant aukštojo lygmens formalizuotus modelius įgalina sumažinti programuotojo darbo apimtį ir greičiau sugeneruoti sistemos specifikaciją atitinkantį kokybišką kodą. Aprašomas mobiliojo roboto valdymo algoritmą įgyvendinančio kodo generavimas naudojant MATLAB terpę. Automatiškai generuojamas kodas lyginamas su žmogaus rašytu programos tekstu. Il. 4, bibl. 5 (anglų kalba; santraukos anglų, rusų ir lietuvių k.).