TEACHING EMBEDDED CONTROL USING CONCURRENT JAVA

Webjørn Rekdalsbakken and Ottar L. Osen
Institute of Technology and Nautical Science
Aalesund University College
N-6025 Aalesund, Norway
E-mail: wr@hials.no

KEYWORDS
Embedded Control, Concurrent Java, Small-scale Model Building, Remote Control.

ABSTRACT

Aalesund University College (AUC) has long and broad experience in the use of the Java Concurrency Model for process control. An important part of this work is the design and control of small-scale embedded systems. These systems include models of many kinds, such as vehicles, motion platforms, pendulums and simulated path tracking systems. The control strategies are mostly based on state space models, using modal and optimal control algorithms. Using realistic small-scale models, software and hardware capabilities and limitations can be tested for possible future full-scale industrial realizations.

INTRODUCTION

Aalesund University College (AUC) has a tradition in the design and building of small-scale embedded control systems. A vital part of the student's Bachelor program in cybernetics has been to develop and build small-scale models of many kinds of dynamic systems, such as pendulums and motion platforms and all kinds of moving vehicles. Often the starting point of a project is a commercially available model building kit, which is being technically upgraded with microcontrollers, motors, sensors and communication hardware to end up as advanced experimental test equipment. An important objective of this activity has also been to involve local industry partners in an innovative process regarding the evaluation of product ideas and potentially further product development. The choice of microcontrollers has been small RISC processors of the types Stamp and PIC. The preferred programming language has become Java, mostly due to the existence of the Java Concurrency Model (JCM) with its simple, yet extensive thread capabilities. The trend has therefore been to choose microcontrollers that have a Java runtime system and a Java Virtual Machine (JVM) included, and with development support for Java. Since 2002 the Java Community Process (JCP) has undertaken a great effort on improving the JCM model with new and improved classes in the *java.util.concurrent* package. Likewise in this period the Real-time Specification for Java (RTSJ) has been revised and much developed under the guidelines of the National Institute for Standardization of Technology (NIST). As a consequence of the work directed by the JCP the Real-time Java (RTJ) model is approaching the requirements of a deterministic real-time system. The development and documentation of the RTSJ standard is the responsibility of the Real-Time for Java Expert Group (RTJEG). The standard is available on the internet at the site of JCP (JCP 2005). These improvements are first being realized on Windows and Linux platforms, but are also gradually being implemented in the embedded microcontroller segment. This development of RTJ is the main argument for using Java in the control and monitoring of hardware equipment, hopefully in a successive progress towards deterministic process control with Java. Being guided by this development in RTJ the cybernetics program at AUC has gained valuable knowledge on using Java in communication and control applications. From the experimentation with small-scale models much experience has also been obtained on the possibilities and limitations of using Java for real-time applications on dedicated microcontrollers (Rekdalsbakken and Styve 2008). This paper is aimed to give a representative presentation of this kind of projects performed by students and teachers at AUC.

THE JAVA CONCURRENCY MODEL

Threads

A thread is an independent process in Java and is the central concept of the JCM. Threads run concurrently and independently within the same JVM. Threads are implemented in Java by the *active object concept* by inheritance from the Thread class. The thread has a *run()* method that executes the thread program. The central mechanism for synchronizing threads is the *monitor*. A monitor encapsulates a shared resource and provides a procedural interface to that resource. In Java the monitor concept relies on a *mutual exclusion* lock for each object. The monitor secures that the procedural access to the lock is executed *atomically*. A method or a block of code must be labelled with the *synchronized* modifier to activate the corresponding object's lock. The first call to the synchronized method will get the lock. Subsequent calls to synchronized procedures within that object will enter the *locked* state and will have to wait for the access. The methods *wait()* and *notify()* have opposite functions; *wait()* is used by a lock owner to temporarily release the object's lock and enter a wait state. This is usually done if a certain condition is not fulfilled. In this way access to the synchronized

Proceedings 23rd European Conference on Modelling and Simulation ©ECMS Javier Otamendi, Andrzej Bargiela, José Luis Montes, Luis Miguel Doncel Pedrera (Editors)
ISBN: 978-0-9553018-8-9 / ISBN: 978-0-9553018-9-6 (CD)

methods of that object is again allowed. A *notify()* called by another thread will alert a waiting thread to regain the lock and continue execution. This thread will normally again perform the test on the vital condition. In this way *wait()* and *notify()* constitute the basic mechanism for the synchronization of threads in Java. A drawback of JCM, however, is the lack of *conditional variables*. These are used to discriminate between the different object locks that threads are waiting for. They will have to be implemented by the programmer. A relative time delay is implemented with the method *sleep()*, and the method *yield()* enables the current thread to give up the execution. This scheme is the basis of the JCM. By these mechanisms all common real-time devices may be implemented, like semaphores, signals, event flags, blackboards etc. The actual implementation of such methods and devices is specific for each microcontroller system. The JCM model is well documented in recent text books (Goetz 2006; Magee and Cramer 2006; Wellings 2004).

The Real-Time Specification for Java

The JCM gives an adequate platform for parallel program execution in Java. It does not, however, specifically support any real-time requirements. The RTSJ is developed on basis of the requirements on Real-time Java set by the US National Institute of Standards and Technology (NIST). RTSJ concerns the implementation requirements of the Java Virtual Machine and also gives specifications for the use of the concurrency model. The crucial property of a real-time system is determinism. In short RTSJ deals with timing requirements to the system's response to critical actions, including specifications for asynchronous event handling and transfer of control. These properties are provided as special classes in RTSJ. These mechanisms are highly dependent on the interaction between system software and the hardware. On a hardware platform supported by a traditional operating system, an application programmer often has little influence on these matters aside from following the recommendations for the use of the concurrency model. On an embedded controller system the run-time facilities are usually much simpler, but the programmer has more freedom to improvise and implement mechanisms adapted to the needs of the situation. The developing process on real-time Java and on the RTSJ standard is found at the web site of RTJ (RTJEG 2005).

Embedded Java

An increasing number of microcontrollers are supplied with an embedded JVM. On the smallest systems only a limited part of the standard Java API is available and there are also restrictions on the use of memory and the number of concurrent processes. The scheduler regimes are also usually quite simple and will vary from system to system. In this paper applications developed on two such small microcontroller systems are presented. One of these is the SBC65EC SBC card from Modtronix built around a PIC controller from Microchip Technology. This controller is furnished with a Java boot loader and a Muvium Developers Kit, including a Java runtime environment and an Integrated Development Environment (IDE). It can be programmed using standard Java, e.g. by us of NetBeans. The other system is the Javelin Stamp from Parallax Inc. Both these systems have a JVM, also called a Java OS, installed. However, both systems API software have limitations compared to standard Java SE. The Javelin has to be programmed using its own IDE. In short the Muvium system has the possibility to run concurrent threads. The scheduler is nonpreemptive, so that each thread has to voluntarily give up execution by a call to *yield()*, and the threads have no priorities. The Javelin Stamp can in principle only run a single thread. It has, however, implemented in firmware a quite extensive system of Virtual Peripherals (VP) to administer the controller's I/O functions. These VPs can either run independently in the background of the application software or in parallel in the foreground. In this way both microcontroller systems can be programmed to perform concurrent processing.

EMBEDDED REAL-TIME CONTROL

Java Microcontroller Boards

The microcontroller market offers several computer boards built upon a microprocessor with a Java interpreter integrated in hardware, or a boot loaded JVM that is tailored to the processor. In this work two microcontroller boards have been explored, the Muvium SBC65EC SBC computer and The Javelin Stamp Demo Board (JSDB). Both systems are equipped with a concurrent Java model and both have an IDE for program development and testing in Java. The benefit of working with small dedicated controllers is the close connection to hardware and hands-on control of I/O and peripherals. The drawback is the lack of or limited access to services from an OS system, but this is often outweighed by the possibilities to tailor the software to the special purpose in question. Also the IDEs for these microcontrollers have been developed to include online execution control and testing of ports and variables. Peripheral facilities have also been much enhanced on the smallest microcontrollers both for analog and digital I/O ports, often including PWM signals and ADCs of high resolution. The communication possibilities are generally quite good. As a minimum the microcontrollers support SSPI. Depending on the microcontroller board support for RS-232 and even TCP/UDP protocols, CAN and Profibus may be available.

The Muvium SBC65EC Development Board

The SBC65EC is a development board from Modtronix built around a PIC18F6627 microcontroller and with a RTL8019AS Ethernet controller and a RJ45 plug for network connection (Modtronix 2006). Muvium is the name of a JVM that can be installed on this controller

with an accompanying Java API (Muvium 2005). This Muvium OS includes a simple scheduler that allows a programmer to run concurrent threads. The SBC65EC is well supplied with peripheral units, among which is a 10 Mbs Ethernet connection with a TCP/IP stack supporting socket connections for TCP and UDP. It has two serial ports with RS232 interface. The board also has 32 general user programmable I/O ports, which can be configured for instance as a 10 bits analog input and a 4 bit PWM output. The Muvium IDE on the PC uses a RS232 port for communication with the SBC65EC. The serial communication is performed by using either of the packages *Javax.com* or *RxTx*.

The Javelin Stamp Demo Board (JSDB)

The JSDB is furnished with an integrated hardware Java interpreter and an internal 16 bits bus for data transfer (Parallax 2006). This assures fast transfer of the byte code between a SRAM and the interpreter and effective execution of code. The controller is richly equipped with I/O ports and communication hardware. These peripherals are controlled by implemented firmware called Virtual Peripherals (VP). These VPs can run either in the foreground or independently in the background of the current thread. Although the runtime system allows only one running user thread at a time, the VPs supply the programmer with quite efficient tools for concurrent execution of necessary processes. The JSDB connects to a PC through a serial RS232 communication port for program download and debugging.

Small-scale Model Building

The microcontroller systems have been used in small-scale hardware models to test their capacities and abilities regarding communication and I/O functions. Many of these models are vehicles based on commercial model kits built to scale, e.g. 1:10. These model kits are modified and further developed by use of all kinds of necessary equipment, like sensors, motors and communication peripherals. The resulting models are then programmed to perform all kinds of conceivable operations, with a view to possibly later realization in industrial products. Examples are tracking systems for light sources or selected objects; automatic topological levelling for a mechanical digger, autonomous radio controlled vehicles, GPS-tracking vehicle, two-wheel balancing devices like wheelchairs, etc. Many such operations may be realistically tested by accurate small-scale models using timely equipment and methods.

TWO-WHEEL BALANCING MONSTER TRUCK

Peripherals

The aim of this project is to make a 4WD monster truck balance on its rear wheels. The car is a HPI Wheely King model, scaled 1:12 of real size. To measure the angle of the car to the vertical axis, an accelerometer of the type Memsic MXD2125G is used (Memsic 2005). This accelerometer uses a thermal principle which also enables it to measure the angle to the vertical line, like an inclinometer. The angle is represented by a PWM signal where a pulse width of 5000 µs represents horizontal position and a ±1250 µs deviation from this represents ±90º. The position and velocity of the car are measured indirectly by integration and differentiation of the arc of the angle of the rear wheels. The vehicle is furnished with two 7.2 V DC motors. To control the speed of the motors a motor controller of the type Parallax HB-25 was used. This controller has a PWM output and includes an H-bridge for direction control.

Control Strategy

To be able to control the system a state space model of the car was developed (Goher and Tokhi 2008). The car is modelled as a uniform rod connected to a circular cylinder (rear wheels). The mathematics is based on physical conservation laws, i.e. Kirchoff's and Newton's laws. The derivation results in the following four-dimensional state space model:

$$\dot{z} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \dfrac{K_e^2\left(m_1 l r - J_1 - m_1 l^2\right)}{A r^2 R} & \dfrac{m_1^2 g l^2}{A} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \dfrac{K_e^2\left(Br - m_1 l\right)}{A r^2 R} & \dfrac{B m_1 g l}{A} & 0 \end{bmatrix} \cdot z + \begin{bmatrix} 0 \\ -\dfrac{K_e\left(m_1 l r - J_1 - m_1 l^2\right)}{A r R} \\ 0 \\ -\dfrac{K_e\left(Br - m_1 l\right)}{A r R} \end{bmatrix} \cdot u \qquad (1)$$

where: $A = BJ_1 + m_1 l^2\left(m_2 + \dfrac{J_2}{r^2}\right)$, $B = m_1 + m_2 + \dfrac{J_2}{r^2}$.

In these expressions the index 1 refers to the body of the truck (uniform rod) and index 2 refers to the rear wheels. The symbol K_e is the electrical motor constant, r is the radius of the rear wheels, and l is the length from the centre of rear wheels to the mass centre of the vehicle. The state variables of the system are defined as follows:

$z_1 = x = \text{position}$, $z_2 = \dot{x} = \text{velocity}$,
$z_3 = \varphi = \text{vertical angle}$, $z_4 = \dot{\varphi} = \text{angle velocity}$

The manipulating signal is $u = U_a = \text{motor voltage}$. Numerical values for all constants are obtained from written specifications or through measurements. By inserting these values into model (1) a system with eigenvalues equal to $\lambda = \begin{bmatrix} 0 & -0.003 & -5.608 & 5.608 \end{bmatrix}$ is obtained. As expected this reveals an unstable system.

The state-space model is the basis for controlling the physical model through the control law, $u = -\boldsymbol{K} \cdot z$. The $\boldsymbol{K}$ vector may be obtained by different techniques. In this case two methods have been explored. The first is the pole placement technique using Ackermann's formula with a Butterworth polynomial forming the characteristic equation, the other is the Linear Quadric Regulator (LQR) with proper weights on the different state variables and manipulation signal. The system band width was estimated to 20 Hz and the $\boldsymbol{K}$ vector was calculated with both methods. Simulations show stable systems for both techniques, but the $\boldsymbol{K}$ values obtained by using the Butterworth polynomial became very large, and had to be scaled down. The LQR method, however, produced a control vector equal to $\boldsymbol{K} = \begin{bmatrix} -44.7 & -78.9 & 6488 & 1162 \end{bmatrix}$, which resulted in the following eigenvalues for the controlled system:

$$\begin{bmatrix} -5.61 + i0.22 & -5.61 - i0.22 & -0.71 + i0.71 & -0.71 - i0.71 \end{bmatrix}$$

This shows a stable system with reasonable damping and response time. The control algorithm based on these values was simulated with good results. The regulator was then implemented in the vehicle. In such experiments much effort on trial and error has to be carried out, but eventually the vehicle was tuned to balance for shorter periods of time.

Software

The software was developed in Java with the NetBeans IDE including the Muvium library for the SBC65EC controller board. Muvium has the capability of running independent threads and includes support for common real-time mechanisms like semaphores, FIFO buffers, blackboards etc. The threads, however, have to explicitly give up execution by calling the *yield()* method. At least one thread has to extend the standard class *UVMRunnable()* to make a runnable program. A picture of the truck is shown in Figure 1, and the UML class diagram for the car software is shown in Figure 2.

Figure 1: Picture of the Balancing Truck

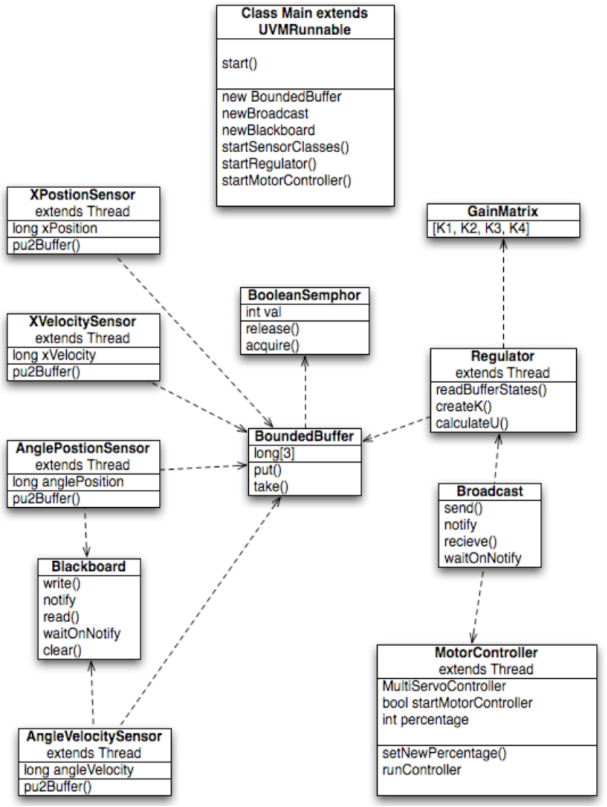

Figure 2: UML Class Diagram for the Balancing Truck

RADIO CONTROLLED TEST VEHICLE

Wiimote Remote Control and WiiUseJ

The objective of this project is to construct a test vehicle that can be remotely controlled from a Wiimote controller from Nintendo. Communication between the Wiimote and PC goes through a standard Bluetooth device on the PC. The incoming information is treated by use of the WiiUseJ Java library (Googlecode 2008). The information from the Wiimote over the Bluetooth connection is gathered by the PC in a data structure. These data are wrapped into a simple protocol that is being transmitted as a telegram over a TCP/IP socket connection to a Muvium controller board. The telegram contains 8 bytes, which is enough to control the remote vehicle. On the Muvium controller board the telegram is forwarded over a serial RS232 line to an onboard radio transceiver chip. The transmitted radio signals are picked up by another radio transceiver placed on a second Muvium controller board, located onboard the test vehicle. Through this communication chain the vehicle gets its control information from the Wiimote controller. The vehicle is supplied with sensors for different kinds of measurements like temperature, light and acceleration, and operates as an autonomous test vehicle sending the sensor information back to the PC. The aim is to further supply the vehicle with a camera and proximity sensors to allow for maximum independent operation, however with possible remote interference from the Wiimote controller over the radio link, see Figure 3 for an overview of the total system.

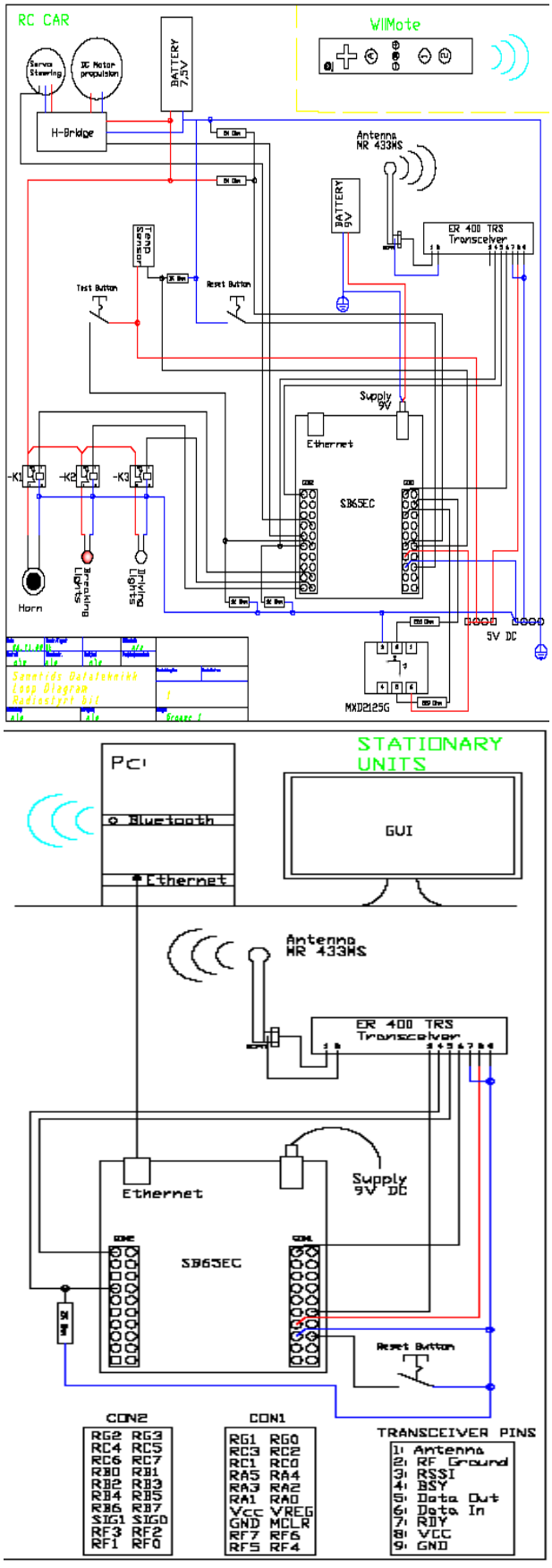

Figure 3: Circuit Diagram for Test Vehicle. Upper Part is the remote Vehicle; lower Part is the stationary PC

Socket Connection with TCP/IP

The Muvium controller board is furnished with an Ethernet connection and a Java.net program package with classes for network communication, included *socket* and *socketService*. With this package a socket connection between the PC and Muvium controller board can be set up in Java, and a TCP/IP protocol may be established. The Muvium board is furnished with a radio transceiver and functions as the remote communication link to the vehicle. On the Muvium radio transceiver a socket connection is established as a server for the PC client. On the PC a *TranceiverHandler* object is defined which sets up the socket connection to the Muvium board on the given IP address, and gets access to the socket's input and output streams. On the PC two independent threads are created; one is reading the data from the Wiimote controller, wrapping them into telegrams and transporting them to the Muvium radio controller over TCP/IP line. The other thread gathers the data that come in the opposite direction from the vehicle via the Muvium radio transceiver, and presents the data in a GUI on the PC.

Radio Communication

The radio communication is established by the use of a RF transceiver of the type Easy-Radio ER400TRS (LPRS 2003). It has a frequency of 433MHz and 10 mW output power. This gives a range of 250 metres in an area with free sight. To improve the range an extra antenna was mounted. The interface is via a half-duplex serial RS232 line.

Sensors

The test vehicle is supplied with several kinds of sensors to monitor its operations and to explore the possibilities for use in practical situations. The main sensors are a Memsic MXD2125G thermal 2-axes accelerometer, and a Phillips KTY81-120 resistance temperature sensor. In addition the battery supply voltage is measured by an analog input port on the Muvium controller, and the radio signal power is read from the transceiver. The accelerometer is used to measure extreme stress situations on the vehicle, e.g. a crash or tip over.

Software

All the software is written in Java using NetBeans IDE. On the PC the WiiUseJ library is used for communicating with the Wiimote remote controller, including the *WiimoteListener* used to capture events from the Wiimote controller. A user-friendly GUI was developed by use of the Java interface *KeyListener*, which enables events to be generated when using the keyboard or the mouse. The SBC65EC controller boards are programmed using the Muvium API delivered with them. On the test vehicle three independent threads are created in the *main* class to run on the controller. These threads have the following responsibilities:

DatagramHandler – receive and control the telegrams from the radio transceiver. Effectuate the commands contained in the telegrams.

SendResponseThread – collect the information obtained by the onboard sensors and retransmit the data to the PC in a proper telegram.

SafetyThread – by use of the accelerometer, monitor the motion of the car and stop its operation if a hazardous condition occurs (e.g. a car crash). A message is sent to the host PC.

These threads use methods in auxiliary classes to effectuate the necessary operations on the car peripherals.

GPS TRACKING VEHICLE

Remote or Autonomous Control

This vehicle is designed as an autonomous path tracking device controlled by a JSDB board and guided by an embedded GPS receiver. The vehicle, however, is also connected to a remote PC through a radio connection. Thus the vehicle can by manually controlled from the PC by a Wiimote controller, or it can follow a predestined path by use of the GPS coordinate information. The radio transceiver is of the type Easy-Radio ER400TRS as described above.

GPS Tracking

The vehicle is furnished with a GlobalSat GPS EM406A receiver (GlobalSat 2007). This is an OEM module with SiRF Star II chipset including an antenna. It communicates on a 4800 baud serial line supporting the NMEA 0183 protocol. It has high tracking sensitivity and a position accuracy of up to 5 metres. The acquisition time is 0.1 sec in average. The GPS is connected to the Javelin stamp's serial RS232 interface, which runs independently as a VP in the background. In the calculations of the car position a starting origo point is defined. On 62° N the ratio of 1° latitude to 1° longitude is 2.16. In the GPS coordinate measurements this ratio is counted for, and the arc tangent function is used to calculate the angle to the next waypoint. In this way the car is following a path by linear extrapolation from waypoint to waypoint. This is not a very precise method because errors in position will accumulate, and therefore a new correct origo has to be determined after some steps. Future improvements in GPS accuracy may give opportunity for a continuous path update towards the waypoint.

Motor Control

Motor control is performed with a Pololu Micro Serial Servo controller (Pololu 2005). This module controls up to eight servo motors through PWM output ports. It connects by TTL logic serial lines (SSPI) both to the JSDB board and to the ER400TRS radio transceiver. In this way the speed and steering of the car may be controlled either from the Javelin stamp onboard the car or from the Wiimote controller on the PC through the radio communication line. Figure 4 shows the circuit diagram for the connection between the radio transceiver and the Pololu controller, and Figure 5 shows the connection diagram between the PC and the radio transceiver.

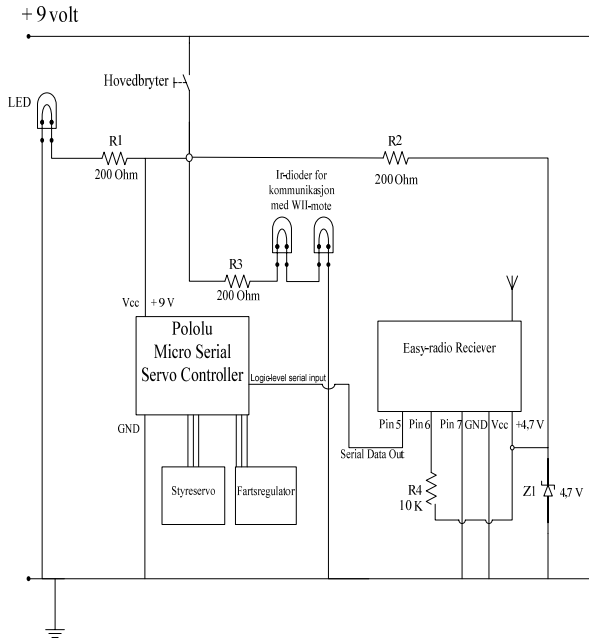

Figure 4
Circuit Diagram for the Connection of ER400TRS and the Pololu Micro Serial Controller

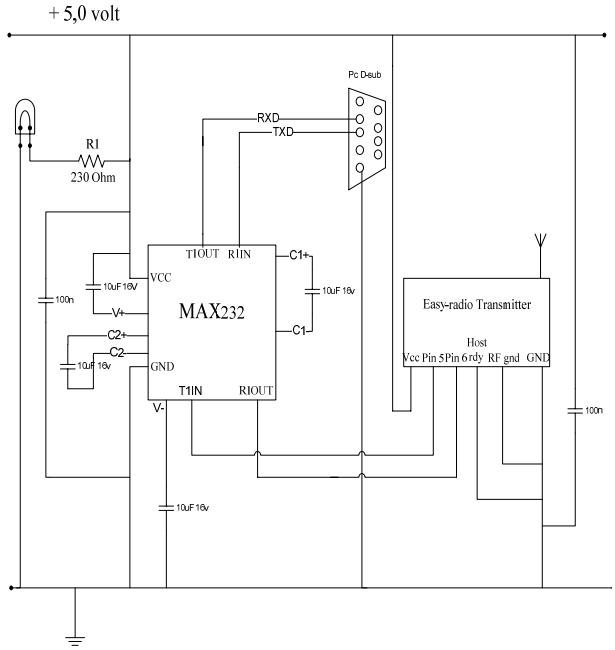

Figure 5
Circuit Diagram for the Connection of the PC and ER400TRS

Software

In this project the software for the autonomous GPS-tracking function is developed in Java for the Javelin stamp processor using the dedicated Javelin IDE and Java libraries. The serial connections from the Javelin stamp to the GPS and Pololu circuits are realized by use of the UART and SSPI VPs on the Javelin. These programs are performing their tasks independently of the main thread. In addition the software consists of an independent Java program running on the PC for the radio controlled steering of the car. This program uses the WiiUseJ library, implementing the *WiimoteListener* to capture events from the Wiimote remote controller. It also uses the (now obsolete) *javax.com* package for serial communication to the radio transceiver. The *javax.com* may be substituted by another serial communication package, e. g. *RxTx*.

RESULTS

These student's projects have revealed that Java is a program language that may be successfully used for process control applications on embedded systems. The great ongoing activity on the development of the JCM model and not least on the RTSJ standard encourages real-time programming with Java. Java is seen to have the necessary real-time mechanisms and also the power for fulfilling the timing requirements in process control. There is, however, still a way to go to guarantee determinism with the JCM. This is a general design problem, and much can be done by building specifically tailored dedicated systems.

CONCLUSION

The model building activity has two main goals; to explore the possibilities and limitations of using Java as a real-time system in embedded process control, and to create a context for experimenting with dedicated microcontrollers and the existing assortment of peripheral circuits in realistic process control and communication applications. The small-scale models have given a context for the testing of communication equipment and software, and the tools for experimenting with different kinds of control algorithms. This experience will improve the understanding of communication and control systems in general, and may give ideas for the building of full-scale devices. The students get a practical approach to control systems and the possibility to test the theories they learn in cybernetics and real-time programming. Finally, the scientific staff is inspired to further explore this technology into new student projects for future challenges. Some of these models may also give offspring into new products in collaboration with local industry partners.

REFERENCES

GlobalSat 2007. Product User Manual GPS Receiver Engine Board EM406A. GlobalSat Technology Corp. www.globalsat.com.tw.

Goetz B. 2006. Java Concurrency in Practice. Addison Wesley. Pearson Education, Inc. ISBN 0-321-34960-1.

Goher K. and M. Tokhi. 2008. "Modelling, Simulation and balance Control of a Two-Wheeled Robotic Machine with Static Variation in Load Position" In *Proc. of ECMS 22th European Conference on Modelling and Simulation*, (Nicosia, June 3-6), 181-187.

Googlecode 2008. wiiusej, Java API for Windows. www.code.google.com/wiiusej.

JCP 2005. JSR 282: RTSJ version 1.1 http://jcp.org/en/jsr

LPRS 2003. Easy-Radio ER400TRS Transceiver LPRS data Sheeet. Low Power Radio Solutions Ltd. www.easy-radio.co.uk.

Magee J. and J. Kramer. 2006. Concurrency. State models and Java programming. Wiley & Sons, Ltd. England. 2006. ISBN-13: 978-0-470-09356-6.

Memsic 2005. Improved, Ultra Low Noise ±g Dual Axis Accelerometer with Digital Outputs. www.memsic.com

Modtronix Eng. 2006. SBC65EC Ethernet enabled Single Board Computer. www.modtronix.com

Muvium 2005. µVMDK muvium Device Development Kit. www.muvium.com

Parallax Inc. 2006. Javelin Stamp Manual, version 1.0. www.parallax.com.

Pololu 2005. Micro Serial Servo Controller. User's Guide. www.pololu.com.

Rekdalsbakken, W. and A. Styve. 2007. "Real-Time Process Control with Concurrent Java." In *Proc. of 6th EUROSIM Congress on Modelling and Simulation*, (Ljubljana, Sept., 9-13), 120.

RTJEG 2005. Documents and Papers. www.rtsj.org/docs

Wellings A. 2004. Concurrent and Real-Time Programming in Java. John Wiley & Sons, Ltd. England. ISBN 0-470-84437-X.

AUTHOR BIBLIOGRAPHY

WEBJØRN REKDALSBAKKEN got his MSc. in Physics at The Norwegian Institute of Technology in 1977. He has been assistant professor and rector at Aalesund Engineering College, and is now assoc. professor and leader of the BSc. programme in Automation at Aalesund University College.

OTTAR L. OSEN is MSc. in Cybernetics from the Norwegian Institute of Technology in 1991. He is a senior instrument engineer at Offshore Simulation Centre AS, and assistant professor at Aalesund University College.