Module 0
Support Vector Machine

ECML/PKDD 2008
http://nxtmote.sf.net
Rasmus Ulslev Pedersen

The presentation overview.

1. Intro to machine learning
2. Classification, Clustering, Regression
3. Implementation issues
4. Support Vector Machine

Overview

Machine learning.

- It is the ability for a machine (ie. computer) to learn from past examples and continually adopt to new situations (information).
- Machine learning is currently a challenge (read: opportunity for research) in wireless sensor networks
- Challenges include battery constraints, radio communication, slow processors (compared to PCs), and limited memory available.

Classification.

- Easy and popular algorithms such *k-NN*, which assigns the same class to a test point as the majority of its n nearest neighbors
- Neural networks were (and perhaps still are) very popular and has been subject to extensive research.
- More recent research is in the area of kernel learners such as the support vector machine.
- Clustering: *K-means* for example
- Regression analysis like least-square-regression in the simplest form

Issues

- The small micro processors does not include a floating point unit, meaning the float data type is a software implementation that is more expensive than usual
- The consequence is that battery, response time or other factors are affected.
- The benefits are of course ease of use and a less error prone implementation

Floating point and fixed point

- Float vs. FP
- FP operations
- Some fixed point operations are easy: addition and subtraction are like we are used to
- Multiplication and division unvolves bit shifting

Bitwise addition:

```
 0001
+0001
 ====
 0010
 ====
```

General Motivation.

- Learning and prediction is/should be an integral part of real-time embedded and distributed systems
- Taking a machine learning approach to intelligent solutions in embedded/distributed systems make design space tradeoffs more rigorous
- A certain class of machine learning algorithms is instance-based and formulated with mathematics/statistics/probability theory (Vapnik)
- Support vector machines are examples of such algorithms
- It is an emerging/maturing field and there are some interesting application areas:
    - Wireless sensor networks: reduce communication
    - Embedded real-time systems: WCET for non-linear predictions

Objective

- Motivate the primary constraints of sensor network nodes
  - Battery
  - Memory
  - CPU
  - Radio
- Many ML algorithms can address constraints
  - Instance based: support vector machines etc.
  - Parametric: expectation maximization etc.
- Tutorial purpose: Use support vector machines as a method to see how one particular class of algorithms maps onto the problem
  - The embedded part
  - The distributed part
  - The (constrained) learning part

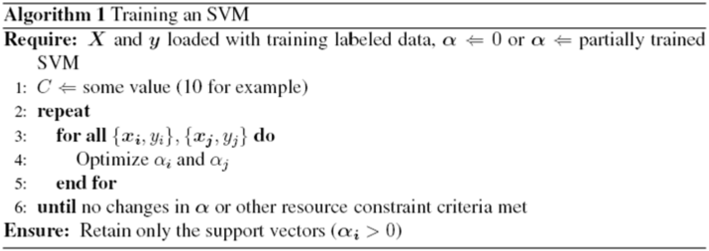
Support Vector Machine

- Linear and non-linear classification
- Instance-based (a good feature for wireless sensors): It allows selected data to be sent around in a sensor network. You can say that the algorithm is somewhat transparent to the existing application
- Instead imagine a neural network or some other model based algorithm. It would probably not be immediately clear which 10% of the data points to send from one node to the next

$$f(\boldsymbol{x}, \alpha, b) = \{\pm 1\} = sgn\left(\sum_{i=1}^{l} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b\right)$$

Training an SVM

- Parameter *C* controls misclassification behaviour/penalty
- Parameter α controls shape of separating hyper plane

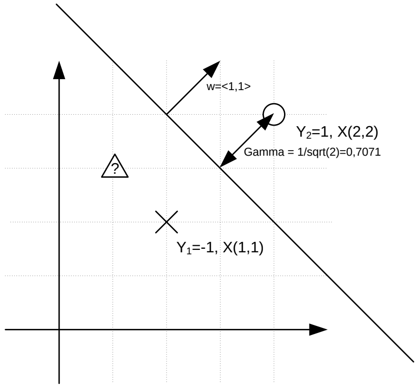


Algorithm 1 Training an SVM

```
Require: X and y loaded with training labeled data, $\alpha \Leftarrow 0$ or $\alpha \Leftarrow$ partially trained
    SVM
 1: $C \Leftarrow$ some value (10 for example)
 2: repeat
 3:     for all $\{x_i, y_i\}, \{x_j, y_j\}$ do
 4:         Optimize $\alpha_i$ and $\alpha_j$
 5:     end for
 6: until no changes in $\alpha$ or other resource constraint criteria met
Ensure: Retain only the support vectors ($\alpha_i > 0$)
```

Margins in an SVM

- Introductory example
  - Binary classification
  - Training data in O and X point
  - Test data in ?
  - Real life test?



⟷	Margin
⟶	A Vector
○	Class 1
╳	Class -1
△?	Unknown Class

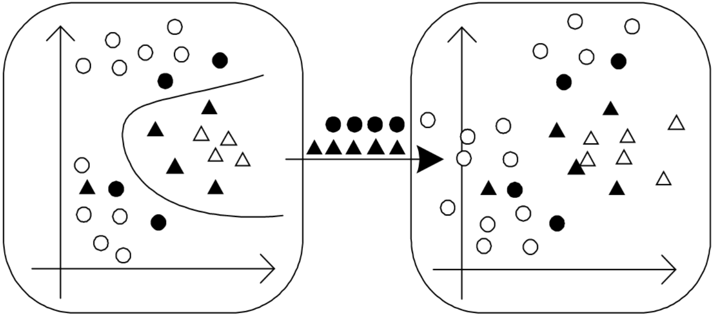
Sequential Minimal Optimization

Pseudo code for the SMO algorithm (for your information only):

1. Outer loop (first choice heuristic): Alternate with a scan of the full data set and multiple partial scans of the non-bound, *NB*, (not 0 or C) data set.
2. For each point—from either scan type—find those that violates the KKT conditions (greater than ϵ), and call inner loop for each such violating point. Terminate the outer loop when all points obey the KKT conditions (within ϵ)
3. Inner loop (second choice heuristic): SMO chooses the second point such that numerator in the calculation of α_2^{new} is likely to maximize the step size. For positive E_1 SMO chooses minimum E_2. For negative E_1, SMO chooses maximum E_2.
4. If no progress is made from first and second choice heuristics then a new non-bound example is searched for, and if that also fails then an entire iteration is started.

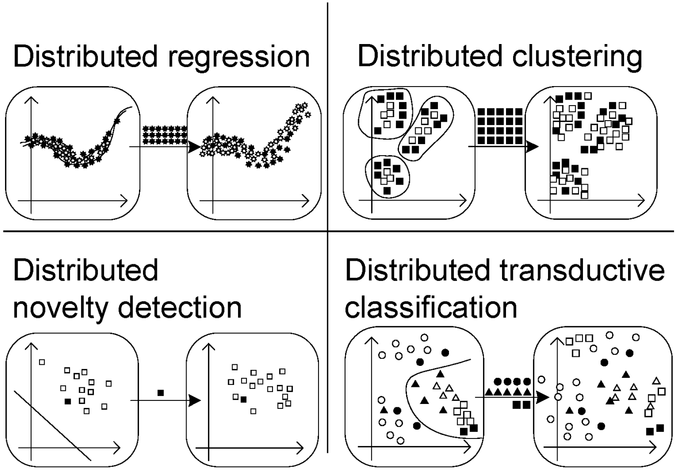
A Simple Distributed SVM

- Distributed classification
- Exchange instances according to some scheme: convex hull, support vectors, max. alpha etc.



More Distributed SVMs

- Distributed regression, clustering, novelty detection and transductive classification
- Exchange instances according to some scheme: convex hull, support vectors, max. alpha etc.


Distributed regression
Distributed clustering
Distributed novelty detection
Distributed transductive classification

Mapping the SVM to the Constraints

- Battery
  - Small 8-bit CPUs are often used...
  - SVM: Local classification can save on radio bandwidth
- CPU
  - 4MHz compared 1500 MHz...
  - SVM: Switch to linear kernel, co-active training
- Memory
  - KB compared to GB...
  - SVM: You can scale from nomem. footprint to full kernel matrix
- Radio
  - Small bandwidth compared to wired networks. Expensive to use radio
  - SVM: Exchange *important* data with neighbors
- Accuracy
  - Hard-real time, soft-real time or ASAP...
  - SVM: Real-time enables on special operation system

Module OE

Support Vector Machine Demo (walk through)

ECML/PKDD 2008
http://nxtmote.sf.net
Rasmus Ulslev Pedersen

The presentation overview.

1. How to use the NXT for machine learning (or more correctly support vector machine)
2. Support Vector Machine. Simple simplified SMO (sequential minimal optimization)
3. Single node

Gliederung

1. TinySVM
2. Training Data
3. Adding data
4. TSVM: Tiny Support Vector Machine
5. BTSVM
6. Extra

The Algorithm.

- Please have a look at the `TestTSVMM` and `TinySVMM` code. Can you see the similarity between the thesis SVM pages and this?
- Can you identify the place where most of the computation takes place?
- Look at thesis called *dsvm-thesis.pdf* around page 30

Early Stop.

- The TinySVM will sample max NUMDATA data points in NUMDIM dimensions
- The left button samples negative instances and the right button samples positive instances
- You can sample 1,2, or 3 dimensional data with the light, sound, and touch sensor.
- In `TinySVMM.nc` you enter training data in the method `prep()`. The setVal method automatically adjusts to the number of dimensions
- Try to make a sample similar to the one in the dsvm thesis
- Try to recompute the alphas. Does it correspond with the results on the screen
- Try to interpret the data on the screen

Early Stop.

- Try to make the SVM run to completion. How long (count kernel evaluations) did it take?
- Now try to enter a third training point? How many kernel evaluations did it take?
- Try to draw a solution on paper and verify it?

SVM Basics

The goal of the exercise is to understand the SVM a little bit.

- Find the SVM directory.
- Take a look at the data structure. You can find it in TestSVMM.nc. It is named `svmdatap` and it is defined in the tosml.h file.
- Try to open the `svm.h` file and see how the things are defined. Then try to open the DSVM thesis on page 18 to see what these variables mean:
- x?
- y?
- a?
- e?
- w?
- b?
- f?
- ob?
- c?
- g?

Sending/Receiving

The goal of the exercise is to send a packets from the master to the slave and let the slave classify the data point.

- Find the BTSVM directory.
- Then open the Makefile in that directory and comment out the line where MASTER is defined.
- Enter some data values in the `prep()` section for the SVM that makes *sense* for you in relation to the sensor readings.
- Now you can compile the slave by writing `make nxtmote` on the command line
- After flashing the slave, then you can read the last two bytes of the BT address. Those two bytes has to be written in the Makefile for the `SLADDR` address. An example is `0xC47A`.
- Then compile the master and flash it. After the search period it will hopefully find the slave and start sending sensor readings to it.
- If you have entered training data correctly then you can create both negative function outputs and positive function outputs. Right?

Extra exercise

- If time permits, you can try to implement it such that the kernel matrix for the training matrix is cached
- Try to create a data matrix in TinySVMM around line 19.
- Make a method called initkernels() and do the computation here.

Module 1
NXTMOTE Project

ECML/PKDD 2008
http://nxtmote.sf.net
Rasmus Ulslev Pedersen

The presentation overview.

1. Program
2. Vision
3. History
4. Structure
5. Links
6. Sourgeforge
7. Contrib
8. Documentation
9. Wiki
10. TODO

Overview

1. **Program**

2. **Motivation**

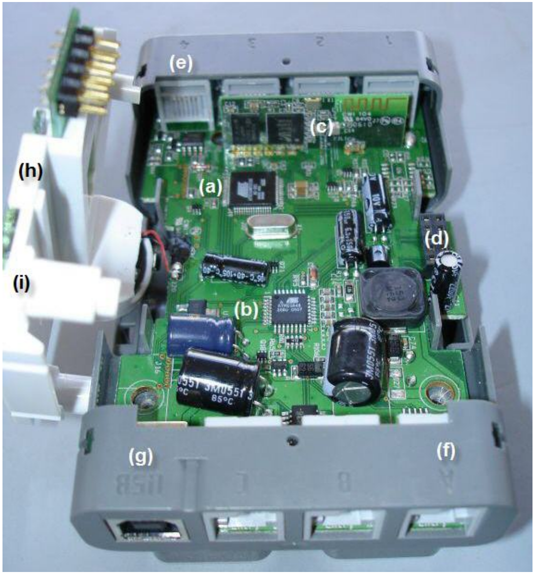
3. **Further Reading**

Program

- Presentations here
- Exercises
- Groups each with 2 Mindstorms. A new exercise will start every approx. 15-45 min and the previous one will be discussed by me

Taking a look at the hardware

Let's start with the fun part...
If we removed the upper part of NXT and looked inside...



The Idea with NXTMOTE.

- To enable a larger group of people to explore what TinyOS is and what it can do
- To raise awareness of different operating system approaches by comparing the firmware replacements for NXT
- To be a flexible platform for embedded systems education and allow experimentation with a multitude of sensor setups

Timeline.

1. 2006: Contact with LEGO and decision to start the project
2. 2006: Convert LEGO NXT IAR compiler code to be gcc compatible and make first "Hello WorldTinyOS on NXT approaches by comparing the firmware replacements for NXT
3. 2007: Setting up various core TinyOS matters like interupts, timers, sensors, initialization, radio
4. 2008: Starting to use NXTMOTE for TinyOS teaching purposes

More complete list at `http://nxtmote.sourceforge.net`.

Structure.

- The platform code is in the CVS repository provided by the TinyOS project. It is called tinyos-contrib. You can find it from the main TinyOS website `http://tinyos.net`
- Documentation is found at the companion website at sourgeforge.net under the name **nxtmote**
- There is a wiki (provided by the TinyOS documentation group) which can be used for a NXTMOTE community to provide project links and additional documentation
- Suggestions, bugs, etc. can be directed to the project e-mail nxtmote@gmail.com

Links.

- There are some links that will be a good starting point
- `http://www.tinyos.net`
- `http://docs.tinyos.net`
- `http://mindstorms.lego.com`
- `http://nxtgcc.sourceforge.net`
- `http://nxtmote.sourceforge.net`

Sourceforge.

How can one contribute to the TinyOS community? Here is an example based on *nxtmote*

- The so-called TinyOS caretakers assure that a well-maintained project stucture for contributed projects is in place
- NXTMOTE is one of these projects and the code is available via CVS
- The other sourceforge project named nxtmote provides access to the releases which are compressed files

Documentation.

The nxtmote project is young still, and you are the first group to really try it. There is some way to go, but here is the intended structure of the documentation.

- The slides provide a first view to the various aspects of NXTMOTE and TinyOS technologies necessary to get started
- The wiki pages co-hosted under the TinyOS docs are dynamic in nature and for use by anyone using NXTMOTE
- The code is documented to some extent and the nesdoc API for NXTMOTE can be browsed from the project home page

Some ideas for you

- There is an idea list on the nxtmote website if someone is up for a project on their own
- Perhaps you: Develop an application and link to it from
- There is a sourceforge project named *nxtmoteprojects* where you can place a copy of nxtmote and start some student project. Someone is starting up on a student bluetooth project from a US university
- You can provide some project information in the nxtmote wiki at `docs.tinyos.net`

Readings

- R. U. Pedersen. Book chapter: TinyOS Education with LEGO MINDSTORMS NXT on NXTMOTE
- As MINDSTORMS NXT is based on an ARM7, I recommend that you go to the Atmel website and read the technical documentation on the processor.
- A first step is to read the TinyOS programming manual, but we will get back to that later

Module 2
Hardware

ECML/PKDD 2008
http://nxtmote.sf.net
Rasmus Ulslev Pedersen

The presentation overview.

1. The main microprocessor in NXT: ARM7
2. The main communication protocols on the board
3. Components of NXT: Display, input/output ports etc.
4. Types of sensors
5. Relation between the coprocessor AT48 and the ARM7

Outline

Hardware Issues

- Startup
- Registers
- Modes
- Display
- Inputs ports
- Output ports
- BT

Startup Process.

- The system starts in an assembler file that sets up the interrupts
- The **main** method is in the RealMainP module
- Different initialization levels and the boot event
- After the boot procedure is done your application component will have the high level code that connect to various event handlers
- Example: If we have a bluetooth application then we will have a method for sending packets and on the receiver side an event that is called when the underlying bluetooth layers receive a packet

Registers, registers, registers...

- The ARM7 can be in a number of different modes
- The registers provide the interface to do all sorts of things with the processor. For example you can set up timers from registers
- Example: `AT91_Timer_Reg_Mode = TRIGGERONMATCH`...
- All registers are described in the PDF manual **AT91 ARM Thumb-based Microcontrollers** which covers the following list of microcontrollers: AT91SAM7S512, **AT91SAM7S256**, AT91SAM7S128, AT91SAM7S64, AT91SAM7S321, and AT91SAM7S32
- LEGO is using the MCU with 64 KiB RAM and 256 KiB Flash memory
- A register is a memory location in the ARM7 where we can read a register value
- Or write a value to the MCU register
- You can think of it as a method call with an argument (if you write) or a return value (if you read). Similar to getset methods in say Java

Registers II.

- An important register in the timer unit is the channel mode register: **TCCMR**
- Page 405 tells us that the TCCMR is offset 0x04 in the timer unit struct
- It is read/write (the write is what we will use)
- it is 32 bits and the first three bits are for selecting a clock to use. The ARM7 has 5 clocks to choose from
- A bit field named *RC Compare Trigger Enable* (**CPCTRG**) depicts if the counter should reset or continue to count when a compare match is encountered

Registers III

```
typedef␣struct␣_AT91S_TC␣{
␣␣␣␣AT91_REG␣␣␣␣␣TC_CCR;␣␣␣␣//␣Channel␣Control␣Register
␣␣␣␣AT91_REG␣␣␣␣␣TC_CMR;␣␣␣␣//␣Channel␣Mode␣Register␣(Capture␣Mode␣/␣Waveform␣Mode)
␣␣␣␣AT91_REG␣␣␣␣␣Reserved0[2];␣␣//
␣␣␣␣AT91_REG␣␣␣␣␣TC_CV;␣␣␣␣␣//␣Counter␣Value
␣␣␣␣AT91_REG␣␣␣␣␣TC_RA;␣␣␣␣␣//␣Register␣A
␣␣␣␣AT91_REG␣␣␣␣␣TC_RB;␣␣␣␣␣//␣Register␣B
␣␣␣␣AT91_REG␣␣␣␣␣TC_RC;␣␣␣␣␣//␣Register␣C
␣␣␣␣AT91_REG␣␣␣␣␣TC_SR;␣␣␣␣␣//␣Status␣Register
␣␣␣␣AT91_REG␣␣␣␣␣TC_IER;␣␣␣␣//␣Interrupt␣Enable␣Register
␣␣␣␣AT91_REG␣␣␣␣␣TC_IDR;␣␣␣␣//␣Interrupt␣Disable␣Register
␣␣␣␣AT91_REG␣␣␣␣␣TC_IMR;␣␣␣␣//␣Interrupt␣Mask␣Register
}␣AT91S_TC,␣*AT91PS_TC;
```

Modes.

- Either the processor is executing the normal program flow or it is processing an interrupt
- The interrupt is triggered by some subsystem on the ARM7 such as the timer
- The ARM7 can branch directly to an interrupt handler based on the subsystem that triggered the interrupt
- There are two interrupt modes: fast and normal
- Important peripheral in this context is the Advanced Interrupt Controller (AIC)
- The AIC will take up to 32 sources and raise prioritized (user-defined) interrupts to the ARM processor

Note that this notion of normal program flow and interrupts carry over to TinyOS and the nesC programming language (as shall be discussed later).

Display.

- The display is 100x64 LCD
- It is controlled via SPI
- It works in a line mode. First a command is sent telling it which line is to be updated then the actual line data is sent
- NXTMOTE API exposes two versions of writing to the display: *Slow* updated that are serviced each ms, and *fast* updates that spin (waiting in a loop) the processor until the display is updated. The latter can be used if the display is used from an interrupt routine
- In the examples that we will look at later, we will see that `sprintf` is used to format a string and a call named `displayString` is used to show the string on the display
- A system timer runs each 1 ms to update the buffer to the display.

Input ports.

- Those four ports that are numbered 1-4 on NXT
- The sensors can be digital or active sensors
- Digital sensors work over a I2C prococol
- Active sensors have an analog value read. The value is then sent back to the ARM7 from the AT48
- Connected to port 4 is a high-speed UART that allows for sensors with higher bandwidth demands (cameras, radios etc.)
- For the exercises we are using the analogue ports from 1 to 3. They provide a value from 0 to 1023.

Output ports.

- The three ports named A-C are output ports
- Each port can provide a PWM signal
- The ports provide feedback the ARM7 to determine rotation direction

Inter MCU communication

- A protocol named I2C is used. It is called TWI (two wire interface) in the Atmel world.
- In NXT is means that a struct is sent from the AT48 to the ARM7 every 2 ms and also from the ARM7 to the AT48 every 2 ms.
- In the struct that comes from the AT48, we find the ADC values from the input ports

Readings

- LEGO MINDSTORMS Hardware and Software documentation. `http://mindstorms.lego.com/Overview/NXTreme.aspx`

Module 3
Software

ECML/PKDD 2008
http://nxtmote.sf.net
Rasmus Ulslev Pedersen

The presentation overview.

1. Main software parts
2. LEGO OS
3. TinyOS parts
4. Low level SW

Gliederung

LEGO.

- AT48 initialization done over I2C from ARM7 (otherwise the ARM7 will be shut down)
- It is a string Let's SAMBA arm in arm...that is sent to the AVR
- One ms update round robin: Display->BT->Sensors->AT48 etc.

TinyOS.

- Timers
- Interrupts
- Scheduler

Timers.

- The timers in TinyOS are defined in the Timer Interface.
- **fired** is the main event in the Timer interface. A number of commands like **startOneShot** and **stop** controls the Timer
- The timer system in TinyOS is fairly complex... It uses almost all of the facilities of the nesC language to provide a virtual parameterized timer interface
- It provides up to 255 individual timers running on top of a few physical timer channels

Interrupts.

- If we start with the hardware, the advanced interrupt controller (AIC) handles the interrupts on the ARM
- Each peripheral (timers, spi, twi, etc) is associated with an id (AT91C_ID_TC0 = 12, AT91C_ID_SPI = 5, AT91C_ID_TWI = 9)
- This ID is used twice when programming an interrupt source in TinyOS. First, it is used when the interrupt interfaces are set up to identify the interface. Secondly, it is used when the interrupt is dispatched to the correct interrupt handler
- An interrupt is assigned a priority
- **nesC** provides language support for tagging interrupt context methods as asynchronous code, which then can use `atomic` blocks to protect state variables

Scheduler.

- Scheduling is in TinyOS is flexible. Out-of-the-box it provides us with a round-robin scheduler that run tasks
- One task does not interrupt another task. Ie. each task runs to completion
- Tasks are defined in a nesC module and *posted* to the scheduler. An error is returned from the posting call if the task is already in queue
- Important: Tasks can post themselves, and this can be useful when working with algorithms that are computationally intensive

Low Level.

- The assembler startup file
- PlatformP (a TinyOS file) that initiates the hardware
- PlatformC which is used to wire to critical components
- In nxtmote we wire to a components that starts the system level timer (PIT timer) to fire each ms

Compilation.

- The compilation process is nesC code to C code and C code to assembler
- The nesc compiler places the C source in app.c and with the correct gcc compiler options we can have a look at the assembler code, corresponding C code statements, and nesC code
- Example: `2059 0bbc 0900A0E3 mov r0, #9`
- It will move the constant 9 into register r0
- 0900A0E3 = 1001 00 0 0000 0 1010 0000 11100011

Readings

- LEGO MINDSTORMS Hardware and Software documentation. `http://mindstorms.lego.com/Overview/NXTreme.aspx`
- You can take a look at `diku.dk/~leopold` and see the TEP 999 draft. It discusses how to set up a new platform if that is part of your plans...

Module 4
TinyOS

ECML/PKDD 2008
http://nxtmote.sf.net
Rasmus Ulslev Pedersen

The presentation overview.

1. TinyOS
2. What is the TinyOS community?
3. How to get started?
4. Conferences and journals

Gliederung

TinyOS Community.

- Main web site at www.tinyos.net
- Mail lists for users and developers
- Core developers from US universities initially, but EU-based developers also represented as well as the rest of the world
- A rapidly growing community...
- Commercial companies are coming up like `http://tinyosmall.com/` and `http://www.xbow.com/`. But do not forget that you can get good generic platform in the form of MINDSTORMS...:-)

Processes.

- TEP: Technical documents that are subject to a public review process. Examples include best practices for development, how to set up a new platform and many more.
- It is also supposed to be the way a contribution makes it way from the TinyOS contribution CVS to the core CVS
- Project listing on `http://www.tinyos.net`
- TinyOS contrib: A sourceforge repository with a process for contributing new projects
- NXTMOTE is one such project
- According to the contrib caretakers there were a huge increase in projects over the last year

What you need.

- Invest in a platform like mica or NXT for example
- There are mote platforms from TinyOS Mall, CrossBow, Intel, Sentialla (upcoming), LEGO MINDSTORMS, Sensinode, BTnut, SUN (Sun Spot) etc. etc.
- There are different operating systems available featuring various languages (nesC, Java, C, BTnut etc.)
- **nesC** provides good structure while still allowing low level HW access (preferred to some)
- If you are new to TinyOS it would make sense to get the LEGO MINDSTORMS platform. The software and hardware documentation provided by LEGO is very good. The NXTMOTE project is an entry-level TinyOS platform with unlimited possibilities due to numerous affordable sensors.

What you can do.

- Do the tutorials. They are now places in the TinyOS wiki
- Read the *TinyOS Programming Manual* a few times
- Read the LEGO documentation
- Read the NXTMOTE manual and slides
- Use nesdoc to browse the code in HTML form
- Make some simple applications
- Sign up for the TinyOS mail lists (tinyos, devel, help, devel, tinyos-2-commits)

Conferences and Journals.

For graduate students and research staff there are numerous possibilities:

- Conferences: TinyOS Technology Exchange (TTX), Information Processing in Sensor Networks (IPSN), Embedded Networked Sensor Systems (SenSys), European conference on Wireless Sensor Networks (EWSN), IEEE Real-Time Systems Symposium (RTSS), International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), International Conference on Body Area Networks (BODYNETS), International Conference on Distributed Computing Systems (ICDCS), International Conference on Distributed Computing in Sensor Networks (DCOSS), International Conference on Networked Sensing Systems (INSS), Workshop on Embedded Networked Sensors (EmNets), etc.
- Journals: ACM Transactions on Sensor Networks (TOSN), International Journal of Distributed Sensor Networks, International Journal of Sensor Networks (IJSNet), etc.
- Other: European Conference on Machine Learning (ECML), International Conference on Machine Learning

Readings

- `http://www.tinyos.net`

Module 5
nesC

ECML/PKDD 2008
http://nxtmote.sf.net
Rasmus Ulslev Pedersen

The presentation overview.

Overview

1. **Introduction**
2. **nesC**
   Interfaces etc.
3. **Further Reading**

Introduction to nesC

- The language using when programming TinyOS
- Associated with some learning curve, but having nesdoc helps.
- nesdoc produce html files based on the nesc files
- It is documented in the nesC language specification and described in the *TinyOS Programming Manual*

nesC.

- Operators: It contains the operators = and ->
- Examples: The operators wire modules together
- Important words: wire, pass-through wiring, modules, interfaces, configurations, parameterized, call, command, event, and signal
- More important words: unique, fan-in, fan-out, provides, uses, implementation

Various nesC code

Examples of pass-through wiring and normal wiring

```
configuration HalBtC {
  provides {
    interface HalBt;
  }
  uses {
    interface BTIOMapComm;
  }
}
implementation {
  components HalBtM, MainC;
  components BTIOMapCommP;
  components BC4ControlC;

  HalBt = HalBtM.HalBt;

  HalBtM -> MainC.Boot;
...
}
```

Interfaces

- nesC interfaces define the contracts for each module
- A interface can be parameterized to some type (advanced use)
- Each function in an interface is defined as an event or a command.
- It is also tagged as async (interrupt context) or sync (default)
- Can be parameterized based on a type

Interface Code

```
interface HalBt {

  /**
   * Send a message.
   *
   * @param msg  Message to send.
   */
  command error_t sendMsg(uint8_t* msg);
...
}
```

Modules.

- nesC programs are composed of components, which are modules, configurations (and components)
- Modules contain the actual code
- Configurations wires it all to one application
- The `implementation` section contain task, command, event or straight C code.

Module code

```
module HalBtM {
  provides interface HalBt;
  uses interface HplBc4;
}
implementation {
  event void Boot.booted() {
    call BTIOMapComm.setIOMapCommAddr(&IOMapComm);
      call BTInit.init();
      cCommInit(NULL);
  }
  command error_t HalBt.sendMsg(uint8_t* msg){
      error_t error = SUCCESS;
    return error;
  }
...
```

Configurations

- Configurations connects the system to become an application
- There are two distinct ways a module work.
- It provides and uses interfaces just a module
- But more importantly it wires (using = or −> in its implementation section) to other modules or configurations

Configuration code

```
configuration HalBtC {
  provides {
    interface HalBt;
  }
  uses {
    interface BTIOMapComm;
  }
}
implementation {
  components HalBtM, ...
  HalBt = HalBtM.HalBt;
  HalBtM -> MainC.Boot;
...
```

Parametrization.

- It is possible that one interface can be used for different types. nesC allows for this.
- The timer interface is defined for various resolutions like milli second
- The parameter value can be used when a *call-back* is issued
- It makes use of the unique and uniquecount words

Default handlers.

- It is a mechanism to make wiring optional for signals or commands
- Examples: The operators wire modules together

Readings

- TinyOS Programming Manual: `http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf`
- nesC Language Reference Manual: `http://nescc.sourceforge.net/papers/nesc-ref.pdf`

Module 6
Sensors

ECML/PKDD 2008
http://nxtmote.sf.net
Rasmus Ulslev Pedersen

The presentation overview.

1. LEGO
2. Mindsensor
3. Hitechnic

Gliederung

Lego Sensors.

- Light
- Touch
- Sound
- Ultra Sound

Standard LEGO MINDSTORMS NXT sensors and NXT-G block

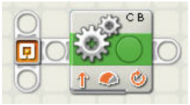
(a) Light

(b) Motor

(c) Sound

(d) Touch

(e) Ultrasound

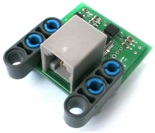


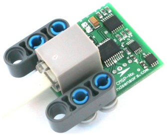
(f) NXT-G block

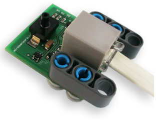
Mindsensors.

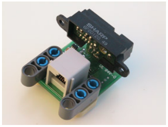
- Camera
- Realtime Clock
- Acceleration (2,3, and 5 axis reading)
- Dual Infra Red Obstacle Detector
- Motor Multiplexer for NXT
- Magnetic compass
- Pneumatic Pressure Sensor
- High Precision Long Range Infrared distance sensor
- High Precision Infrared distance sensor (short, medium, long range)
- Various accessories: port splitter, cables, plugs, sensor kit, tools

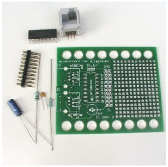
Selected NXT sensors from Mindsensors

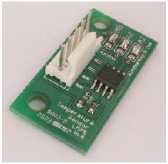

(g) Multi-axis acc.


(h) Magnetic compass


(i) Pneumatic pressure


(j) Infrared distance


(k) Sensor building kit


(l) Temperature

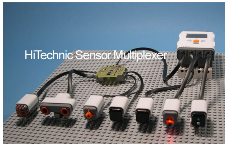
Hitechnic.

- Prototype Board
- Gyro
- IR seeker
- Compass
- Color sensor
- Acceleration/tilt
- Sensor and motor multiplexer
- Accessories: Cables

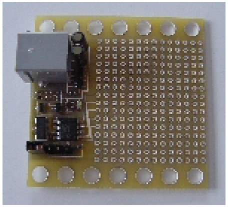
Sensors from Hitechnic

(m) Color

(n) Gyro



(o) Multiplexer


(p) Prototyping board

Sensors for exercises.

- Light sensor
- Touch sensor
- Microphone

Readings

- LEGO MINDSTORMS NXT Hardware Developer Kit.pdf and appendices 1-8.

Module 7
Radio

ECML/PKDD 2008
http://nxtmote.sf.net
Rasmus Ulslev Pedersen

The presentation overview.

1. Radio: Wireless Sensor Networks
2. Standard NXT radio: Bluetooth
3. Zigbee
4. Active messaging

Overview

1 **Wireless Sensor Networks**

2 **NXT Bluetooth**

3 **Bluetooth Module**
NXTMOTE Bluetooth

4 **Further Reading**

WSN Radio Comm.

- The idea is to use wireless communication as a replacement for cables
- Similar to a cabled IP network we have a number of motes sending packets over the air to one another
- Routing protocols exist within WSN networks as they do within IP networks
- Conservation of energy plays a role in WSN
- Dynamic (unpredictable) topology is a characteristic of WSN

Active Messaging.

- A concept to dispatch messages in a network or parallel computer system
- A message id acts as a simple routing mechanism
- To be very specific: If you remember the nesC term *parametrization* then an interface can be parameterized according to an active message id

NXT Bluetooth.

- LEGO NXT uses a Bluetooth radio
- Bluetooth is a wireless short-range standard for communicating between small devices
- In its basic forms a small network with one master and a number of slaves
- For NXT, the Bluetooth radio forms a network with three slaves and one master
- Bluecore (the bluetooth radio) in NXT support the serial port profile

BT module.

- The Bluetooth module in NXT is probably the most complicated piece of software
- From a systems point of view the ARM7 chip controls the Bluecore chip
- A set of communications commands such as starting an enquiry (the way Bluetooth devices discovers neighbors)
- The Bluetooth module has its own software implementation (done by LEGO) running and it serves as the middle-man between two ARM7 processors on two different NXTs
- Raw data is transmitted in what is called ßtream mode"
- We use the TinyOS message abstraction, but in essence we write bytes to the Bluetooth radio

NXTMOTE BT.

- The term *Active Message* is well-developed in TinyOS, and therefore the NXTMOTE BT offers a simplified version.
- Most (99% or so?) of NXTMOTE work with be done within a NXTMOTE network, so we can boil the 6 byte bluetooth address to the 2 bytes address used in the TinyOS active message stack
- A fairly safe approach would be to use a checksum of all the 16 bit parts of the BT address
- But simplicity is easier for most purposes, so we just use the first 16 bits of the BT address
- As the äctive messageïs about to be sent, we find the full BT address of the receiver and send it

Readings

- TEP 111: message_t

Module 8
Exercises

ECML/PKDD 2008
http://nxtmote.sf.net
Rasmus Ulslev Pedersen

The presentation overview.

1. NXTMOTE
2. Hardware
3. Software
4. TinyOS
5. nesC
6. Sensors
7. Radio

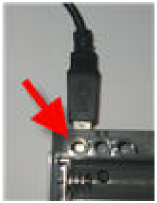
Overview

Preparing the PC

- Install cygwin
- Find the hadat.zip presentations at Sitescape LEGO folder
- Install Textpad
- Place the nesC.syn file in the Textpad systems folder
- Open Textpad and look at the files
- Rename c:/cygwin and unzip the one from LEGO folder to be the default
- Rename c:/cygwin/home/rup.inf to be the user-name of your machine
- Open Cygwin and change to directory $NMA
- Change into the SVM directory and write `make clean` and `make nxtmote`
- Look at the files being created

Preparing NXTMOTE

- Put the sensors into NXT. Light sensor in port 1, touch in port 2, and leave the mic alone for now. You can use it later
- Put the battery in NXT and reset it. Resetting is done by pressing a little button
- Install the NXTMOTE USB driver when NXT is reset (see picture)
- The USB driver is in `C:/cygwin/home/nxtmote`
- In the SVM directory, write `fwflash.exe build/nxtmote/nxtmote.bin`



Registers

- Find the Timer/Counter block on page 4 of the AT91 manual (doc6175.pdf). How many timer channels are there?
- Look at the bullet Compare RC Trigger". Where do we set the RC value?
- The RC value is set in `tos.chips.at91.timer.HplAT91OSTimerM from the file tos.chips.at91.timer.HalAT91AlarmM`. What is the value?
- The channel control register (see page 405) is set with `AT91C_BASE_TC0->TC_CMR = TC_CLKS_MCK2|AT91C_TC_CPCTRG`. The | operator combines the two constants, but what does the constant `AT91C_TC_CPCTRG` mean?

Build System

- Find the file named PlatformP and open it. What is going on in this file?
- Find and analyze the build script named `nxtmote/support/make/at91/at91.rules`.
- What happens in the target named exe0?

Project overview

- Look around at `www.tinyos.net`. When you find the projects overview, try to think which projects can be inspirational for your own endeavors?
- See the tinyos-2.x directory (a few levels back from the nxtmote directory on your PC). Can you find the Timer interface (in some subdirectory)? What is the command to start a one-shot timer?

nesDoc

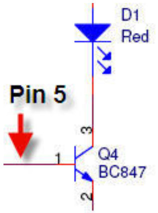
- Try to change into the BTSVM directory and write `make nxtmote docs`
- The documentation is placed in .../nxtmote/doc/nesdoc/nxtmote
- Try to browse it. Where is the BTSVM application?

Modules

- The file nxtmote/tos/chips/at91/timer/HplAT91OSTimerM.nc contains timer code. Which interfaces are provided? Used?
- Use the nesdoc html to see it
- What is the sequence of events when an interrupt fires and is dispatched by the fired event? Try to follow it in the code.
- It starts in `irqhandler()` ...nxtmote/tos/chips/at91/HplAT91InterruptM.nc so you can look there

Light sensor

- Please find the light sensor schematic PDF. It is called `Appendix 3-LEGO MINDSTORMS NXT Light Sensor hardware schematic.pdf`. Can you see what happens when we set pin 5? How does this relate to the `togglepin(1)` debug function in
- Also find the main schematic and find the two processors: ARM7 and AT48. Can you follow the analog input from the light sensor to the AT48 and then see the TWI (I2C) connection between the two processors?
- What is the struct called that holds the ADC light value (see the LEGO HW manual page 20).



Source: LEGO

Sending/Receiving

- What happens in the message header file `nxtmote/apps/tests/bt/BtRadioCountToLeds.h`?
- Sent and received messages are routed internally based on the active message id. Where is that found in the header file?
- `nxtmote/apps/tests/bt/TestBtM.nc` is the module. Which methods are used when sending a packet? And receiving a packet?

Bluetooth radio overview

The goal of the exercise is look at the Bluetooth radio.

- First try to read a little bit about the BT radio in the LEGO pdf files.
- In the file HalBt.nc you should look at the contents.
- Look at the hardware specification. Can you see how the BT radio talks to the ARM? For example try to locate the pin that sets one of the modes?
- Then look carefully at the `cCommUpdateBt()` method.
- Try to map the states of this state machine to the LEGO pdf on the Bluetooth radio API.
- Can you see what the contents of the message between the ARM and the BT module is for the `MSG_GET_LOCAL_ADDR` message?
- Now open the `BtRadioCountToLeds.h` file. It shows how the message is structured.
- Try to make a slave by commenting out the `#define MASTER`

Bluetooth radio application

- Try to make a slave by commenting out the `#define MASTER` in `BT/TestBtM.nc` around line 48
- Start the slave
- Note down the last two bytes. If it says 5C.F then you write 0x5C0F in the TestBtM.nc at line 167
- Also de-comment the `#define MASTER` line
- Start the master

Bluetooth radio cont.

- Compile and load on a slave
- Now find the slave address in the code where the master looks: First line in `task void sendit()`
- Challenge: If you want to then try to send a variable (same type as the counters) along with the message. Can you get it to display on the other device? Hint: Change it in the `BtRadioCountToLeds.h` file and in sending and receiving places in TestBtM.nc.

Controlling the Slave Display

- The format of the output is on one of the 8 lines. The lines are numbered 0-7.
- Look at code in the code in the `Receive.receive(message_t* msg, void* payload, uint8_t len)` code in the TestBtM.nc module.
- The sensor readings are named xv1, xv2, and xv3.
- The formatting of the string managed with sprintf (see `http://www.cplusplus.com/reference/clibrary/cstdio/sprintf.html`)
- Try to display the product (multiply) the two readings
- Try to format it so sensor is *Light:* and the other line is *Touch:*

Module A
NXTGCC Project

ECML/PKDD 2008
http://nxtmote.sf.net
Rasmus Ulslev Pedersen

The presentation overview.

1. NXTGCC
2. LEGO
3. LEGO community

Overview

NXTGCC.

- NXTGCC is based on the LEGO source code. It was provided by LEGO to facilitate the GCC compiler use.

LEGO IAR.

- LEGO uses the IAR compiler internally.
- It is a nice and easy programming tool to use. IAR also provides configurations for some Texas Instruments Zigbee chips.

LEGO Community.

- The LEGO community includes a small group of software/hardware minded people who do firmware replacements
- The open source nature of NXT makes this particular feasible (as opposed the real"hacking that was taking place for the LEGO RCX)
- Main LEGO MINDSTORMS website: http://mindstorms.lego.com/
- Popular sites are: lugnet.com, nxtasy.org and many more

Readings

- NXTGCC note:
  `http://nxtgcc.sourceforge.net/nxtgcc.pdf`