Signals, Instruments, and Systems

School of Architecture, Civil and Environmental Engineering

EPFL, SS 2018-2019

http://disal.epfl.ch/teaching/signals_instruments_systems/
Signals, Instruments, and Systems – W1

Part I: Course Organization, Team, and Content
Team beyond this course

Distributed Intelligent Systems and Algorithms Laboratory: http://disal.epfl.ch

• Instructor: Alcherio Martinoli
• Guest lecturers: Ali Marjovi, Alicja Wasik
• Teaching assistants:
  – Alicja Wasik (Head TA, PhD student)
  – Cyrill Baumann (TA, research assistant)
  – Chiara Ercolani (TA, PhD student)
  – Faezeh Rahbar (TA, PhD student)
  – Sophie Chalumeau (Help TA, master student)
• Support staff:
  – Duarte Dias (Postdoctoral Fellow)
  – Anwar Quraishi (PhD student)
Course Rationale, Content, and Prerequisites
Typical Field Instrumentation for Environmental Monitoring

- Ultrasound anemometer
- Laser-based disdrometer
- Integrated compact weather station (temperature, humidity, anemometer disdrometer)
- Data logger
Local Climate Monitoring

**Features:**
- Very low sampling frequency < 1Hz
- Very low power consumption: 25mW
- Solar panel
- Radio communication

**Sensors:**
- Air Temperature and Humidity
- Infrared Surface Temperature
- Anemometer
- Solar Radiation
- Pluviometer
- Soil moisture
- Soil pressure

Visit [http://www.sensorscope.ch](http://www.sensorscope.ch) for further details!
Ecosystem Monitoring

Science
• Understand response of wild populations (plants and animals) to habitats over time.
• Develop in situ observation of species and ecosystem dynamics.

Techniques
• **Data acquisition of physical and chemical properties, at various spatial and temporal scales, appropriate to the ecosystem, species and habitat.**
• **Automatic identification** of organisms (current techniques involve close-range human observation).
• Measurements over long period of time, taken *in-situ*.
• Harsh environments with extremes in temperature, moisture, obstructions, ...

Source: D. Estrin, UCLA
Embedded Systems at the Heart of Modern Environmental Engineering

- Moving the lab to the field is “in”!
- Most of the applications require large spatial distributions (scale of the domain $\gg$ scale of a node) → sensor networks
- The underlying hardware/software technology (at the single device level) share the same principles
What These Systems Share at their Core?

- Sensing
- Processing
- Mobility
- Communication
- Processing
- Visualization
- Storing

In-situ instrument → Transportation channel → Base station

The goal of this course is to shed light on this process and blocks!
What This Course Is About

• Fundamentals of computer science:
  – Basics of computer architecture
  – C programming consolidation (vs. Matlab)
  – C for embedded and real-time systems

• Fundamentals of signal processing:
  – Analog/digital signals, sampling and reconstruction
  – Time/frequency domains and transforms
  – Filters, converters

• Fundamentals of embedded systems:
  – Microprocessors, microcontrollers, memory
  – Sensors and actuators
  – Basic control and communication techniques and concepts
Course Prerequisites

- Fundamentals in C and Matlab programming
- Fundamentals in probability calculus
- Fundamentals in analysis (Analysis I to IV, ODE, Fourier series and analysis, transforms)
- The Introduction to Control of Dynamical Systems course can help
- Possibly ICC also can help with fundamentals in signal processing
Rationale (1)

- This course must allow you to become a **power user** of the **key instruments** in environmental engineering **used nowadays** (sensor networks, meteorological stations, data loggers, etc.) and **in the near future** (drones, remotely operated vehicles, robotic sensor networks, etc.)

- Being a **power user** means not only being an advanced user but also understand enough to collaborate with EE/CS/ME engineers to **design** the instruments of the future in environmental engineering; idea: **complexity shifted more and more from hardware to software** → a lot of **domain knowledge** can be transferred to the instruments at **software level**
Rationale (2)

• **Well-balanced course**: theory, algorithms, tools and practical exercises

• It should prepare you to better follow a number of master courses (especially in the new Environmental Monitoring and Modeling specialization)

• This course can be considered as an **elevator** to the master course “Distributed Intelligent Systems” for SIE students where various other sections and programs attend

• It should get you prepared for carrying out a design/semester/master **project** at DISAL
Organization of the Course
Credits and Workload

• 5 ECTS
• 1 ECTS = 30 h workload → 150 h workload total
• Rough breakdown
  – 60 h lecture (including reading, exam preparation)
  – 45 h lab (including verification test and its preparation)
  – 45 h course project (including implementation, reporting, reviewing and defense)
Grade

• Final written exam: 180 minutes
• 50% performance during semester, 50% performance during the exam (compromise US/Europe style)
• During semester: lab verification test (20%, tested material of 6 labs, about 30 h effort) and course project (30%)
• During final exam: all covered material is subject to examination, but the exam is open book (no electronic equipment allowed other than a basic, non programmable pocket calculator)
Lecture Notes & Reading

• Policy: no manuscript, but slides and your own notes

• Preliminary lecture slides in pdf format available for download on the course web site before each lecture (wed evening), definitive ones after lecture (couple of days max); e-mail notification only when definitive slides are posted

• Reading will be added on Moodle as the course progresses

• Access to Moodle: in principle you should be all set because compulsory course and therefore inclusion by default; if issues contact sis-ta@groupes.epfl.ch
Tools used in Exercises and Course Projects

- C programming environment
- Matlab
- High-fidelity robotic simulator (Webots)
- Real devices (desktop robots: e-puck; desktop sensor nodes: Arduino kit)
- OS: UNIX (Linux)
e-puck Robot (Labs & Projects)

Computation and memory

Communication

Actuators

Sensors

I/O
Webots (Labs & Projects)

- High-fidelity simulation
- Discrete sensor and actuators
- Noise and nonlinear characteristics faithfully reproduced
- Different trade-offs faithfulness/computational cost
Arduino Kit – V0 (One Lab) (under development)
Labs (1)

• Lab session 15:15-18:00 on Tue, GR B0 01 (first lab session: next week)
• Mini-tutorial (5-10 min) by the main lab designer at the beginning of the lab
• 10 lab sets total
• Official solution available on Moodle after the lab session
• sis-ta@groupes.epfl.ch for ANY issue!
Labs (2)

- Lab assignment posted by Monday of the same week at latest

- Lab presence not compulsory but
  - We do not repeat labs
  - If you do not come to labs you will have a hard time in the lab verification test and being efficient in the course project
  - Certain labs involve HW which we do not distribute outside the lab sessions

- I think it is a lot of fun and really helps you understand lecture stuff

- Assistants are well prepared (2 TAs, designer and tester, and help TA)

- **Dedicated feedback forms** for each lab on Moodle
Lab Verification Test

- Worked well in the last 5 editions of our courses
- In Week 8 (concerns content of Lab 1-6)
- 3 h duration, during exercise session (in the computer room)
- Graded and individually reviewed (no official solution)
- Note: content of Lab 7-10 will be exploited in the course projects and verified in the final exam
Suggestions for a Successful Exercise Series

From our teaching experience:

• Read the assignment in advance (i.e. before the lab), in this way you will be more efficient when the TAs are around for helping you on the toughest questions…

• Take advantage of office hours (upon appointment using the sis-ta list) if additional explanations are needed

• Do the labs seriously, do not wait for the solution distributed a couple of days later.
Course Project (1)

- Course project list distributed in Week 5, kick-off in Week 8
- 45 h effort, defenses first week after the end of the semester, Wed June 5 (1.5 h presence max, can pick the time slot).
- Team of 3 students (ad hoc solution for numbers of students not divisible by 3)
- Will distribute HW/SW at home
- Short progress report (compulsory but not graded) in Week 10
- Final report to be submitted (pre-established max # of pages and format) by end of Week 14 (Sun June 2)
Course Project (2)

• Final presentation in front of the class (10 minutes + 10 min questions)
• Each of the project will have another team of students as reviewers
• Each of the team members has to present
• Pre-established regular office hours:
  – 1 h kick-off
  – 1 h each week for each of the different topics (Week 9 to 12)
  – Week 13 and 14: 3 h of the lab session
Suggestions for a Successful Course Project

**From our teaching experience:**

- Exploit office hours for the course project in an efficient way (e.g., ask the toughest technical questions but do not ask the project supervisor to debug your code!)
- Plan your effort (milestones, time, constraints, etc.), coordinate your team
- Arrive at the progress report milestone with project objective understood, reading and tool familiarization over, and preliminary implementation results
Final Notes
11th Iteration at ENAC

• Same organization and content as last year; only minor improvement when possible
• e-puck in good health and compatibility under current Linux OS ok (tested in DIS).
• Mica-z replaced by an Arduino kit (kit and lab under development)
• Major BS reshaping on-going, substantial changes will be introduced for the course edition in two years from now (2020-2021); this and next edition, no major changes
Signals, Instruments, and Systems – W1

Part II: C programming refresher
Linux Operating System

• Interface between hardware and applications. Manages and coordinates activities and sharing of the limited computer resources
• Standard shell commands (ls, cd, pwd, cp, mv, rm, mkdir, man…)
• Absolute and relative paths
• Environmental variables ($PATH, $HOME)
• Redirection of input and output
• Pipes

Source: http://en.wikipedia.org/wiki/Operating_system
Remember `man`?

• You can use `man` to show information about functions in the standard libraries of C:
  – e.g. `man printf`
  – or `man atan2`

• To type in the terminal!
C compared to interpreted languages (e.g. Java, Python)

- C is NOT object oriented (C++ is OO), while most of the interpreted languages are OO.

- Code is directly translated (through a **compiler**) into binary that can be directly executed by the machine, while the interpreter languages need a running engine (e.g., JVM for Java codes) to be translated to binary (**usually at runtime**). Therefore with C, the programmer can directly access machine resources (memory, processes).

- It’s not ‘safe’, while most interpreted languages are ‘safe’.
Main differences with Matlab

• Matlab is also an interpreted language
• Matlab is optimized for matrix operations
• Syntax differences (loops, functions, etc…)
• No variable declarations
Compilation

```
int main() {
    int a = 5;
    double b = 4.3;
    return a * b;
}
```
The C Compiler Pipeline

Source File
main.c

Header Files
#include stdio.h
#include common.h

C Preprocessor

Object Files
main.o
module.o

Library Files
libc.so

C Compiler

Linker

Executable File
main
“.h” vs. “.c”

• Usually header files (“.h” files) should contain all the necessary functions, structures, typedef and enum declarations such that another programmer can use your code without having to look at your c file.

• C files contain the actual implementation and “hidden” declarations.
Libraries

- Libraries provide special functionality in the form of collections of ready-made functions:

**Library:**
- `stdio.h`
- `math.h`
- `time.h`
- `stdlib.h`

**Example:**
- `printf(const char* format, ...)`
- `sqrt(double x)`
- `gettimeofday()`
- `rand()`

**Usage:**

```c
#include <stdlib.h>
#include "my_library.h" : your own collection of function declarations
```
Compilation Example

Main source file

3 main.c \rightarrow main.o

gcc help.o file.o main.c -o program -lm

Complementary file

1 help.c \rightarrow help.o

gcc -c help.c -o help.o

2 file.c \rightarrow file.o

gcc -c file.c -o file.o

program

libc
standard C library

libm
math library
Makefile: Example

(see previous slide)

```
CC = gcc
LDLIBS = -lm
all: main

main: main.o help.o file.o

clean:
   rm -rf *.o main

- compiler
- additional library
- targets
- label
- [TAB] !!

Note: Run make clean all for a totally new compilation
```
Variables

• There are several types of variables in C:
  – Integers: char, short, int, long
  – Floating point: float, double
  – Characters: char
  – There is no Boolean nor String type.

• Variables can be either signed or unsigned:
  – Ex.: unsigned int, signed char
# Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage size</th>
<th>Value range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1 byte</td>
<td>-128 to 127 or 0 to 255</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1 byte</td>
<td>0 to 255</td>
</tr>
<tr>
<td>signed char</td>
<td>1 byte</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>int</td>
<td>2 or 4 bytes</td>
<td>-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>unsigned int</td>
<td>2 or 4 bytes</td>
<td>0 to 65,535 or 0 to 4,294,967,295</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2 bytes</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>long</td>
<td>4 bytes</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4 bytes</td>
<td>0 to 4,294,967,295</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage size</th>
<th>Value range</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>4 byte</td>
<td>1.2E-38 to 3.4E+38</td>
<td>6 decimal places</td>
</tr>
<tr>
<td>double</td>
<td>8 byte</td>
<td>2.3E-308 to 1.7E+308</td>
<td>15 decimal places</td>
</tr>
<tr>
<td>long double</td>
<td>10 byte</td>
<td>3.4E-4932 to 1.1E+4932</td>
<td>19 decimal places</td>
</tr>
</tbody>
</table>
Variables

• Variables need to be declared before they are used.

• In correct ANSI C89, variables need to be declared at the beginning of a block (after “{”).

```c
int main() {
    int a = 5;
    double b = 4.3;
    return (int)((double)a * b);
}
```

Remark: A variable type can be modified with a cast.
Variables - Cast

- The cast operator is an operator which forces a particular type mould or type cast onto a value:

```c
char ch = 'a';
int i;
i = (int)ch;
```

```c
int i = 1;
int j = 3;
double k;
k = i/j;
```

```c
int i = 1;
int j = 3;
double k;
k = (double)i/(double)j;
```

- Careful!
Variables - enum

• Enumeration of tags. The tags are numbered (0,1,..) by default.

Def.:  
\[
\text{enum } \text{identifier} \{ \text{enumerator-list} \}
\]

Ex.:  
\[
\begin{align*}
\text{enum BOOLEAN} & \{} \\
\text{false} & = 0, \\
\text{true} & \quad \text{// =1} \\
\}; \\
\text{enum BOOLEAN is_empty} & = \text{false};
\end{align*}
\]
Variables - typedef

• Typedef allows you to define your own types.

Def.: \texttt{typedef type typedef-name}

Ex.: \texttt{typedef int BOOLEAN; BOOLEAN is_empty = 0;}

Controlling the execution flow

- Algorithms are all about controlling the execution flow.
- Algorithms are all about deciding how to proceed depending on some conditional statements.

```c
float max = 0.0;
float a = 5.0;
float b = 2.1;

if (a > b) {
    max = a;
} else {
    max = b;
}

return max;
```
Conditions

- Conditions can be expressed using logical expressions:
  
  >  (greater than)
  <  (less than)
  >= (for greater than or equal to)
  <= (for less than or equal to)
  != (not equal)
  == (to test for equality)

- In C, there is no boolean variable (true or false). Instead, true is represented by any value not equal to 0 and false is represented by the value 0.

```c
int a = 0;
if (a == 1) {
    // this code is reached
} else {
    // this won’t happen
}
```

Do not confuse `a == 1` (equality) with `a = 1` (assignment)

```c
int a = 0;
if (a = 1) {
    // this code is reached
} else {
    // this won’t happen
}
```

Wrong

Correct
Conditional branches

- The **switch** structure is very useful when the execution flow depends on the value of a single integral variable (int, char, short, long).

```java
switch (a) {
    case 1:
        {
            // if a == 1, do this
            break; // jump to the rest of the code
        }
    case 2:
        {
            // if a == 2, do this
            break; // jump to the rest of the code
        }
    default:
        {
            // otherwise, do this
        }
    // rest of the code
}
```

```java
if (a == 1) {
    // if a == 1, do this
} else if (a == 2) {
    // if a == 2, do this
} else {
    // otherwise, do this
}
// rest of the code
```

Both codes have exactly the same behavior!

**Do not forget the **break** instructions, otherwise the statements in the rest of the **switch** will also be executed!**
Conditional loops are a combination of “if..then” and a jump.

```c
int total = 0;
int i = 0;
int array[3] = {12, 3, -5};

while (i < 3) {
    total += array[i];
    i++;
}

return total;
```

What is the value of total at the end of the program?

**total = 10**
Conditional loops

The loop `for` is useful when an iteration count (`i` in the example below) needs to be maintained, but the number of iterations must be known.

```c
int total = 0;
int i = 0;
int array[3] = {12, 3, -5};

for (i = 0; i < 3; i++) {
    total += array[i];
}

return total;
```

What is the value of `total` at the end of the program?

`total = 10`
The statements `break` and `continue` cause the program to exit a loop or to jump directly to its next iteration, respectively.

```c
int total = 0;
int i = 0;
int array[3] = {12,3,-5};

while (1) {
    if (i < 3) {
        total += array[i];
i++;
        continue;
    } else {
        break;
    }
    // unreachable code
}

return i;
```
stdin - stdout

- Standard input and standard output are file streams
- `printf("Hello")` is equivalent to `fprintf(stdout,"Hello")`
- `scanf("%d", &i)` is equivalent to `fscanf(stdin,"%d", &i)`
- `stderr` is useful to display errors, because it is not affected by redirection (`>`): 
  ```
  ./my_program > out.txt  -- the errors will still be displayed!
  ```
Printing – Format

Most commonly used:

- `%d` integer
- `%u` unsigned integer
- `%f` double
- `%s` string
- `%c` char

<table>
<thead>
<tr>
<th>Object</th>
<th>Control spec.</th>
<th>Actual output</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td><code>%6d</code></td>
<td>42</td>
</tr>
<tr>
<td>‘z’</td>
<td><code>%3c</code></td>
<td>z</td>
</tr>
<tr>
<td>2.71828</td>
<td><code>%10f</code></td>
<td>2.71828</td>
</tr>
<tr>
<td>2.71828</td>
<td><code>%10.2f</code></td>
<td>2.71</td>
</tr>
<tr>
<td>“printf”</td>
<td><code>%s</code></td>
<td>printf</td>
</tr>
</tbody>
</table>
Indentation

• Indentation and spacing helps you and others read your code.
• It has to be systematic and consistent.

```c
int main() {
    int i, j;
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            if (i == j){
                printf("%d\n", i);
            }
        }
    }
    return 0;
}
```
Complete Example

- Module for representing, adding and subtracting complex numbers
- Manual compilation vs. Makefile
- Compilation errors
- See the archive on Moodle
Conclusion
Summary

• Basics in Linux:
  – How to navigate, how to manipulate files
  – How to combine commands
  – Where to look for information (man pages)

• Refresher on C programming:
  – Compilation
  – Variables and main function
  – Basic control structures (if, case, while, for)
Additional Literature – Week 1

• Nice summary on linux shell usage: http://linuxcommand.org/learning_the_shell.php

• Classical book for C programming:
  ‘C Programming Language’, Prentice Hall
  Brian W. Kernighan, Dennis M. Ritchie

• Popular C link:
  http://www.c-faq.com/

• And many more – Google is your friend