Achtung! Alles Lookenspeepers!

Das computermachine ist nicht fuer gefingerpoken und mittengrabben. Ist easy schnappen der springenwerk, blowenfusen und poppencorken mit spitzensparken. Ist nicht fuer gewerken bei das dumbkopfen. Das rubbernecken sichtseeren keepen das cotten-pickenen hans in das pockets muss; relaxen und watchen das blinkenlights.

Course Contents

- Concurrency
- Reliability
- Real-time

Discussion of real-time requires an understanding of concurrency

Robustness > Predictability > Performance

(almost) all of you will be programming, not all of you will be working with real-time constraints

Learning Platforms

- Lectures
- Exercise lectures
- Exercises
- Project
- Books, online & other

Project

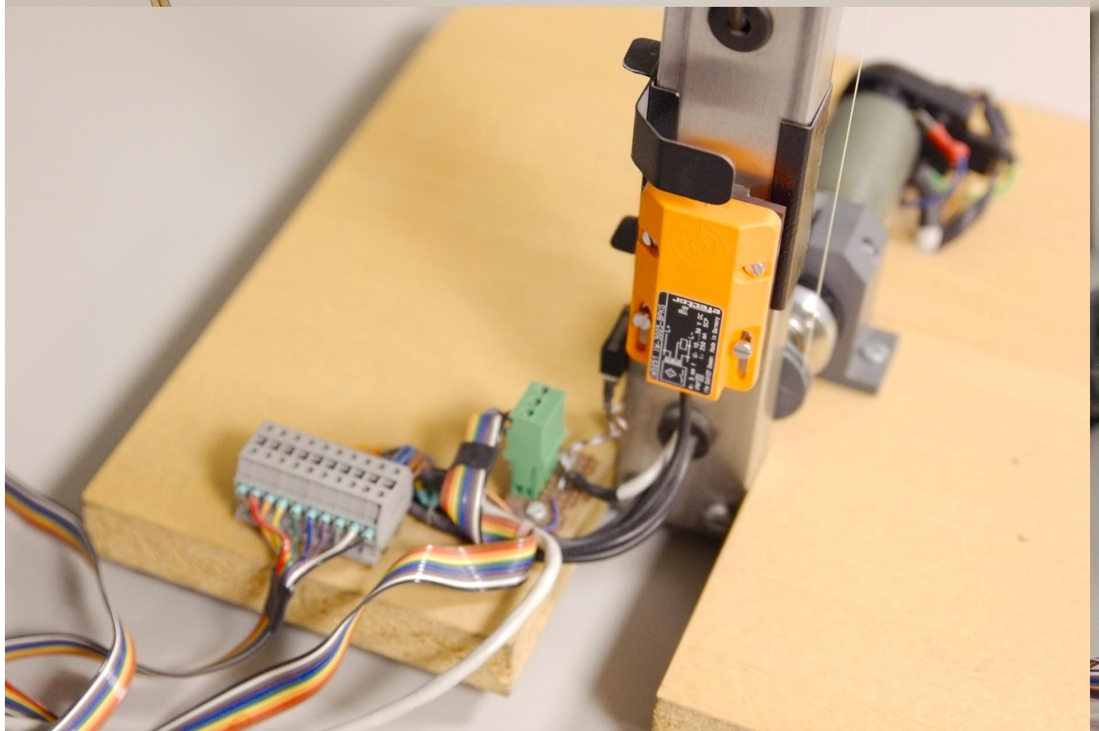
n elevators across m floors working in parallel.

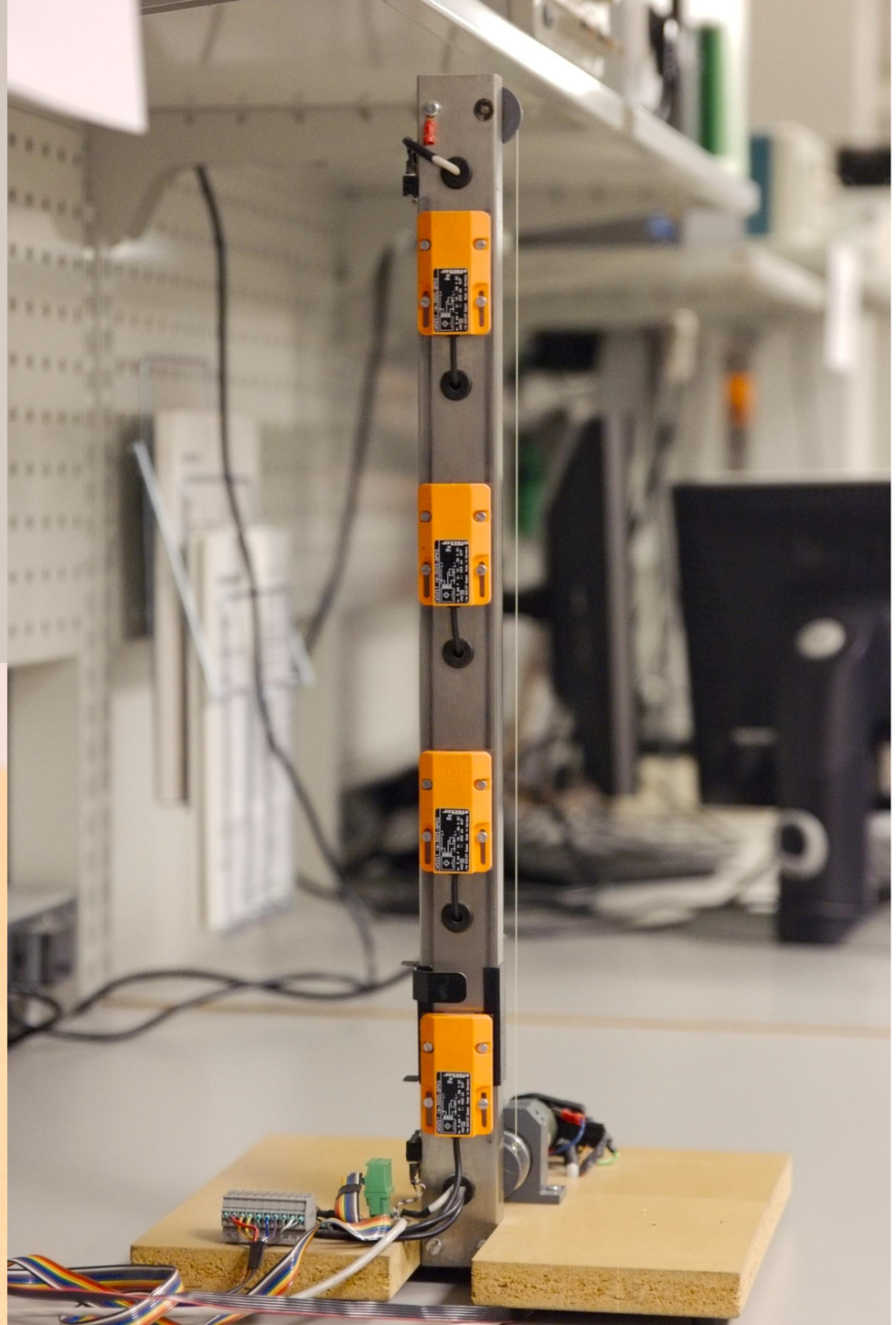
Eg Sentralbygg 2, where n=2 and m=14

Any language

We support C/POSIX, Go, and Python


Etasjepanel
PC
PLS
4 ETASJE
NED
3 ETASJE
NED
OPP
2 ETASJE
NED
OPP
1 ETASJE
OPP
STOPP
DØR ÅPEN
OBSTRUKSJON
AKT
Heispanel
1
2
3
4





Goal:

- No orders are lost
  - An order is “in the system” when the button light is turned on
- All orders are completed within reasonable time
  - If an elevator crashes/dies/becomes a giraffe, another should take over
- Multiple elevators should be more efficient than one
- A single elevator should behave sensibly and efficiently
  - Don't stop at every floor without checking for passengers, in an infinite loop
  - Call upward and call downward should behave differently

Simplifications you are allowed to make:

- At least one elevator will always be alive
- Stop/Obstruction can be ignored
- Start with 1 <= n <= 3 and m == 4
  - Try to avoid hard-coding
  - You do not need to test for other values, but your code should scale
- Communication is reliable
  - Only do this if you are having *real* trouble with the network programming

Reason: something that works > number of features

Underway:

- A few milestones to make sure you all get started:
  - Creating a design outline
  - Designing and implementing the network module
  - Interfacing with the hardware
  - Running a single elevator
- Ask both the assistants and other students for help and ideas
  - Learning by doing only works if you aren’t stuck
  - Experience-based learning gives you the test first and the lesson after

Evaluation:

Completion:

The elevator system actually works

Design:

No orders are lost, the system is robust

Code review:

Do you understand your own code?

Is it easy to add the features you simplified away (if any)?

Are your APIs complete, so you can swap out modules/extend features?

Is your style consistent?

Is your code idiomatic (wrt. the language)?

Exercises

- Hands-on experience with the concepts in the course
  - (At least some of them)
- The project is only one language
  - Some techniques and methods are best demonstrated with a specific language
- Widen your knowledge of the different approaches that exist
  - How can you know why one way is better if you've never learned another way?


javascript

Choosing a language

We can divide the act of programming into three stages:

1. You idea … (the thing you want to create)
2. … translated to …
3. … a formal framework (a programming language)

Most programming courses molest #2
in a misguided attempt to teach #3

You are given an idea (#1) and a language to formalize it in (#3)

Here is a box. Think inside it.

Formalities of a language are a means to an end, not a goal in itself

(Unless you are taking a pure language course)

Programming is more than syntax and semantics

From the bottom up:

- The compiler does “context-free” translation of code to machine instructions
- The programmer does “context-sensitive” translation of specification to code
- The designer/engineer does translation of problem to solution
  - From abstract business requirements to specifications

#2 will be much easier if your choice for #3 suits your mental model of #1

Choose a language that:

- Mirrors/enables you design/solution
- Does not have quirks that annoy you
- That makes you say “well that’s convenient” and “Wow, that's awesome.”

These all depend on your personal way of approaching a problem

Languages that we support:

- C/POSIX
- Go
- Python

Chef’s special this semester:

- D

Other languages you should check out:

- Matlab w/ Stateflow
- Rust
- Java
- Scala
- Erlang
- Ada

C / POSIX

Type	System, imperative, structured, static weak typing
Memory	Manual
Concurrency	None: use POSIX on linux (not cross-platform)
Reliability	Return values, errno, setjmp/longjmp
Network	None: use platform libraries (which are in C, usually)
Other	“Portable assembler” , ubiquitous, void* hell, “easy to learn, hard to master”

Go

Type	Application, imperative, concurrent, string static inferred typing
Memory	Garbage collection
Concurrency	Built-in (goroutines), channels, select, synchronous or buffered, Race-condition tester included
Reliability	Return values (via multiple return values), defer, panic/recover (discouraged), unit tests
Network	Multiplatform, old/netchan (?)
Other	No inheritance (arguably good), no generics, small language Use message passing (channels)!

Python

Type	Application, imperative, OO, functional, dynamic weak typing (optional type identifiers), VM
Memory	Garbage collection
Concurrency	“Multithreading” (GIL), multiprocessing (fork with mmap)
Reliability	Exceptions, with()
Network	Multiplatform, recv_into()
Other	Library heaven, slow

D

Type	Application/system, imperative, OO, functional, meta, generic, strong static inferred typing
Memory	Garbage collection, manual (emplace), void[] composable
Concurrency	Multiplatform, asynchronous message passing, TLS
Reliability	Exceptions, scope(), DBC, unit tests, @safe, pure
Network	Multiplatform, vibe.d
Other	CTFE, mixin, unlimited nesting, template constraints, IFTI, UFCS, large language

MATLAB / Stateflow

Type	Domain-specific, imperative, OO (!), array, dynamic weak typing
Memory	Garbage collection
Concurrency	Parfor, Task, Trigger, #include generated code
Reliability	Exceptions
Network	Some built-in, call java code, MEX, #include generated code
Other	Very visual, cumbersome multilanguage setup

The rest

Rust	No null ptrs, lightweight tasks w/MP, no shared memory
	Young (0.8), steep learning curve
Java	OO only, VM, discourages concurrency
Scala	Java for people who like functional, concurrency, types
	Don’t dive in on the deep end: you’ll drown
Erlang	Concurrent functional, hot code swapping, restart individual tasks
	Pure functional is not for everyone
Ada	
Others?	