ASEBA

Open-Source Low-Level Robot Programming

Stéphane Magnenat [1]
stephane at magnenat.net

Philippe Rétornaz [2]

[1]Autonomous Systems Lab
ETH Zürich

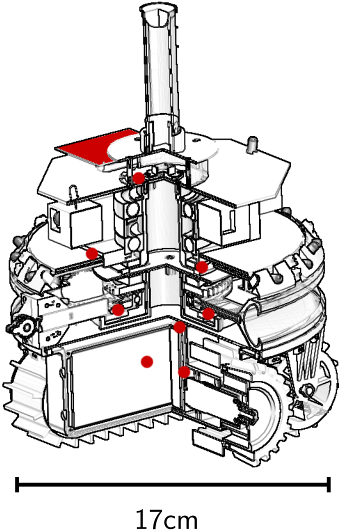
[2]Mobots group - Laboratory of robotics Systems
EPFL

February 5, 2012

Outline

Outline

Motivation: Multi-Microcontrollers Robots



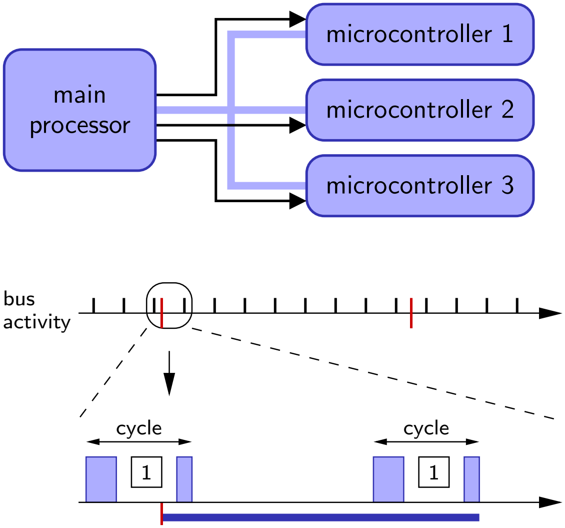
Modern integrated mobile robots have

- lots of degrees of freedom
- many and various sensors

Their computing infrastructure consists of

- a main processor running Linux (ex. Gumstix)
- multiple microcontrollers
- a common communication bus (ex. I2C or CAN)

Motivation: Usual Approach is Centralized Polling...



main processor

continuously

- read sensors
- process data
- set actuators

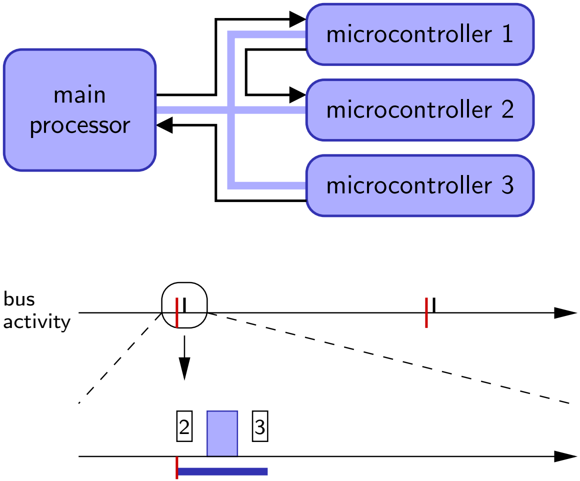
microcontrollers

- interface to physical devices

bus

- I2C

Motivation: ...but Distributed Events are Better!



main processor

- react to events
- send events

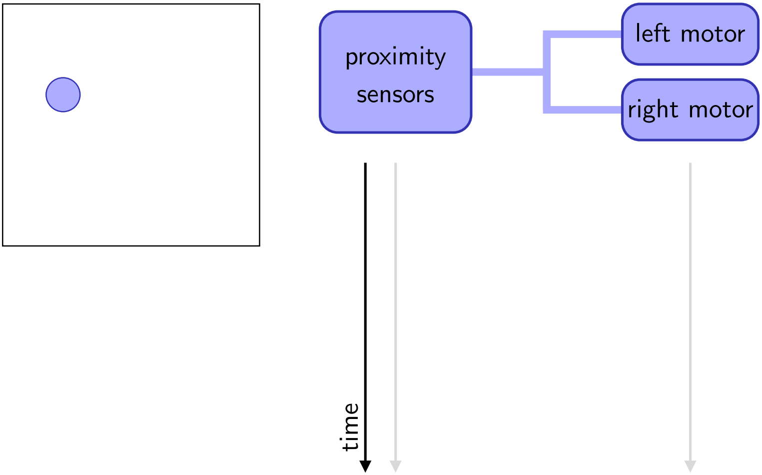
microcontrollers

- interface to physical devices
- preprocess data
- send events
- react to events

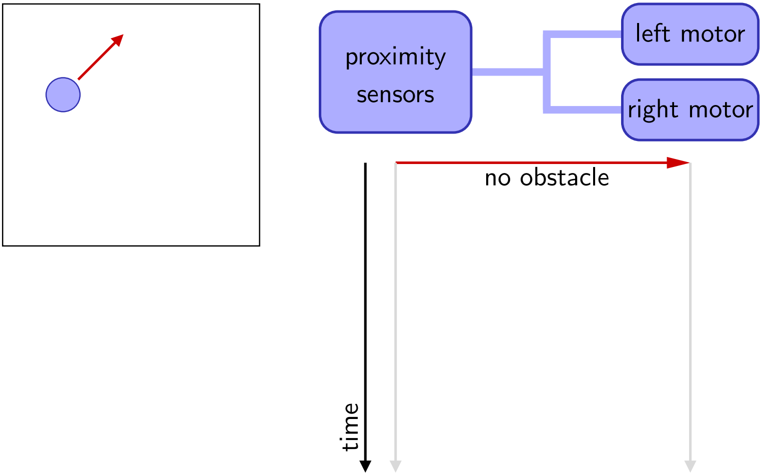
bus

- CAN

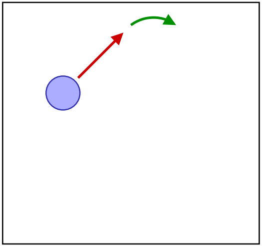
Motivation: An Example of Distributed Events

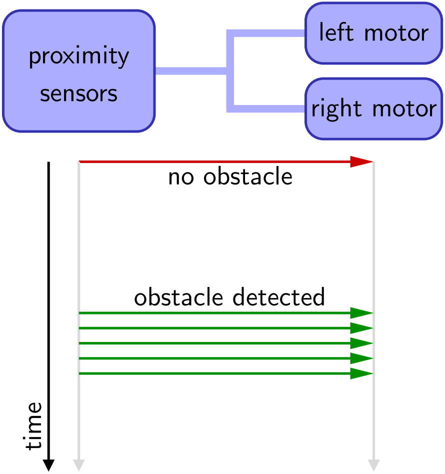


Motivation: An Example of Distributed Events

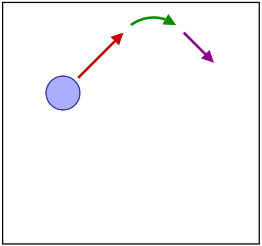


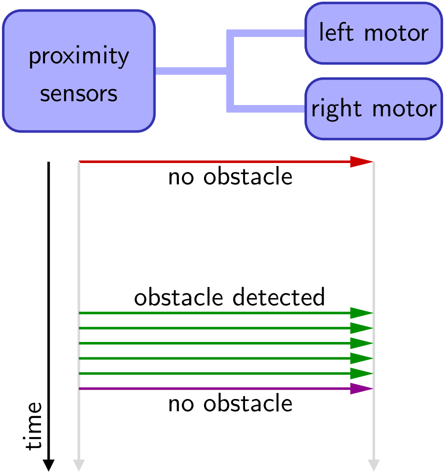
Motivation: An Example of Distributed Events





Motivation: An Example of Distributed Events





Motivation: How to Implement Distributed Events?

- Contrary to centralized polling, with events the emission policy must be distributed as well.
- Microcontrollers must take decisions about what event to send when, and how to link incoming events to actuators.
- Therefore they must be programmable, but flashing is slow.
- A virtual machine is the solution...
- And so Aseba was born!

Motivation: How to Implement Distributed Events?

- Contrary to centralized polling, with events the emission policy must be distributed as well.
- Microcontrollers must take decisions about what event to send when, and how to link incoming events to actuators.
- Therefore they must be programmable, but flashing is slow.
- A virtual machine is the solution...
- And so Aseba was born!

- Aseba stands for Actuator and Sensor Event-Based Architecture.
- Aseba puts virtual machines inside microcontrollers; enabling their programming through a user-friendly language and IDE.

Outline

Swarm Robotics: hand-bot

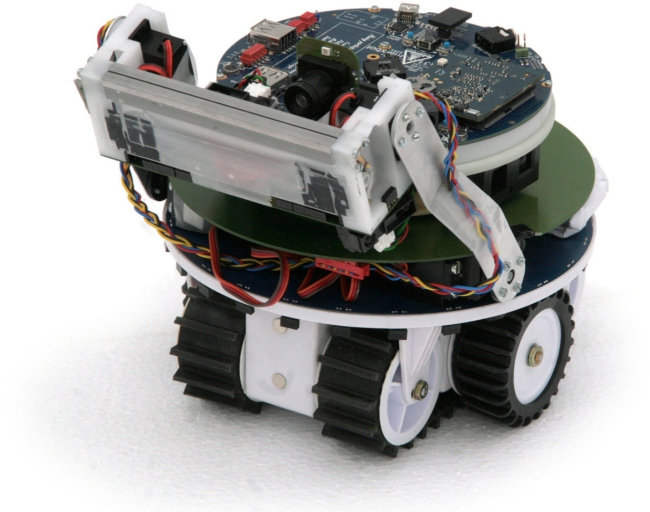
6 Aseba VM, real-time unwind control, grasping, and state machines implemented in Aseba

Swarm Robotics: hand-bot

hand-bot video at
`http://www.youtube.com/watch?v=92bLgE6DO2g`

In this video, the hand-bot climbs a shelf and retrieves a book solely using Aseba.

Autonomous Construction: Lonelybuilder



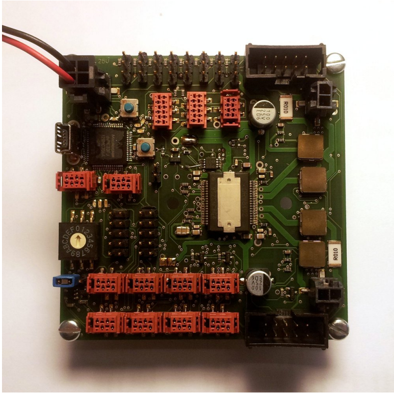
5 Aseba VM, state machines coded in Aseba
localisation, mapping, planning running on Linux ARM

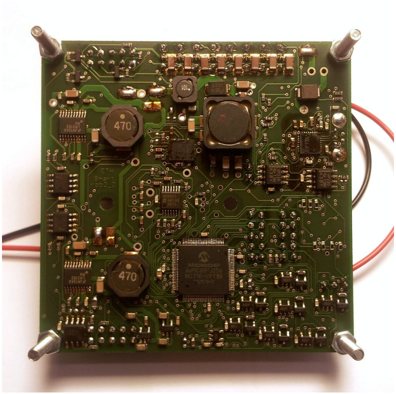
Autonomous Construction: Lonelybuilder

Lonelybuilder video at
`http://www.youtube.com/watch?v=h865RHbT9Ms`

In this video, Lonelybuilder constructs a tower by manipulating cubes. The manipulation state machines are implemented with Aseba.

Brick for Building Robot: Smartrob





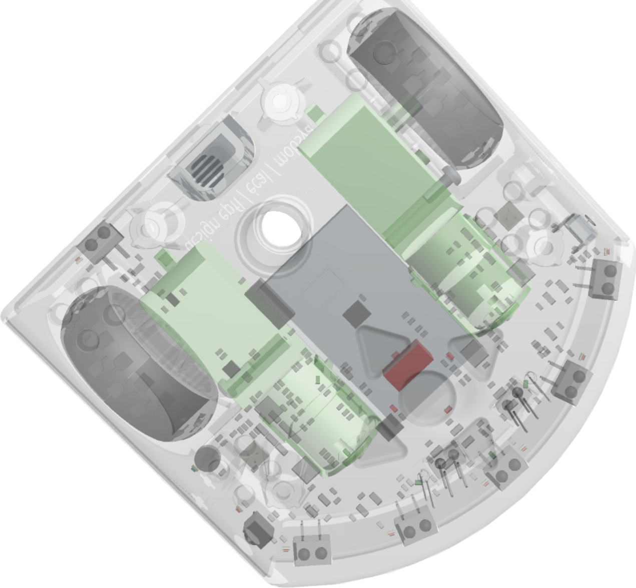
- 2 motor and 8 servo drivers
- 8 infrared-sensor drivers
- additional I/O and A/D
- CAN bus, stackable

- single 3 to 25 V input
- shipped with Aseba
- ROS and D-Bus integration
- soon available for buying

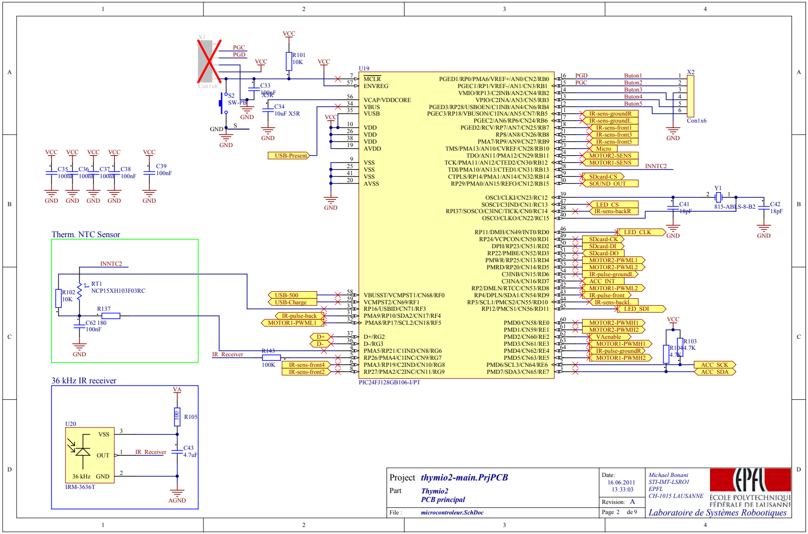
Educational Robotics: Thymio II

Full-featured open-hardware programmable mobile robot for 100 CHF (≈ 80 €), see `http://aseba.wikidot.com/en:thymio`

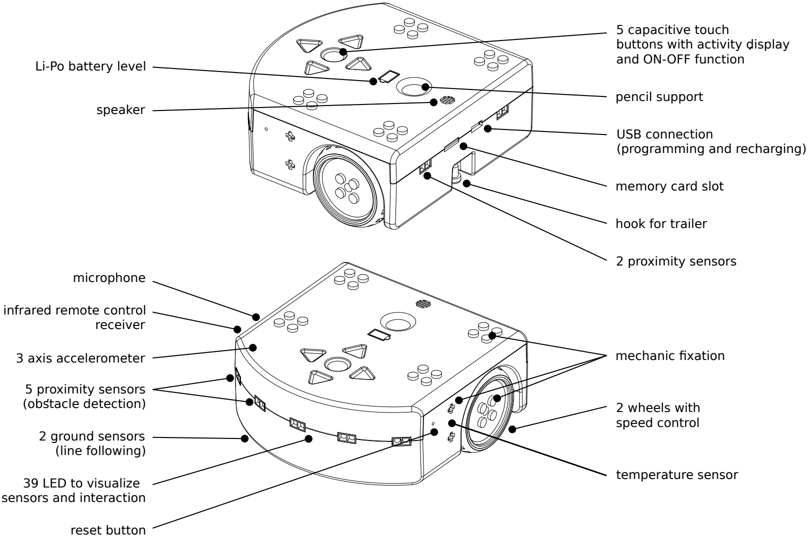
Educational Robotics: Thymio II



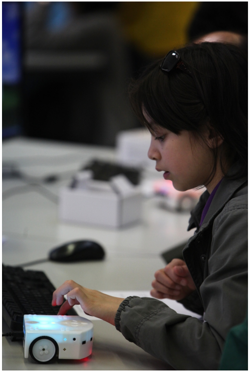
Educational Robotics: Thymio II



Educational Robotics: Thymio II

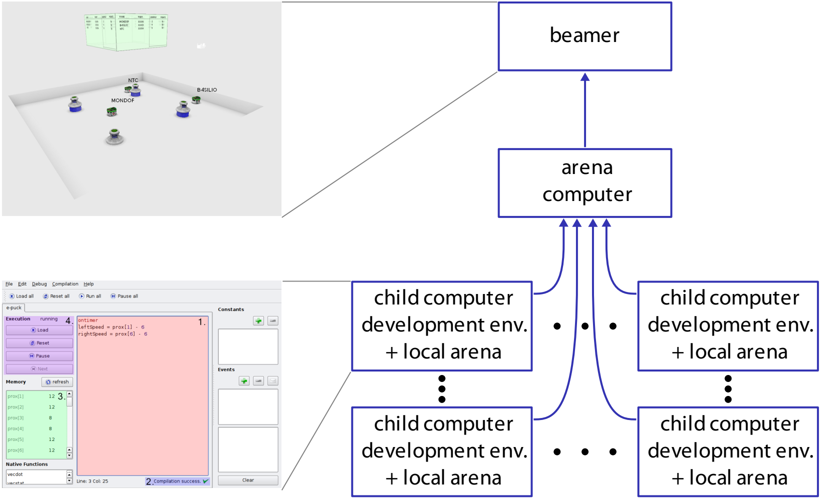


Educational Robotics: Thymio II



Enables children to discover programming with a mobile robot.

Simulated Robotics: Challenge



Outline

Motivation

Current Use

Technical Description

Performances

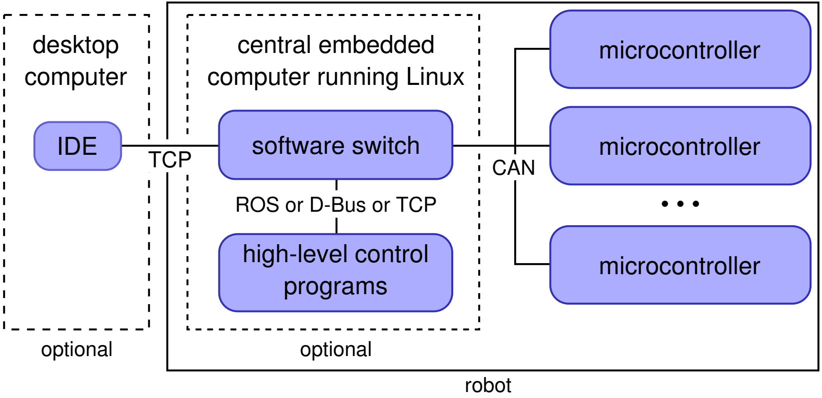
Wrap-up

Technical Overview

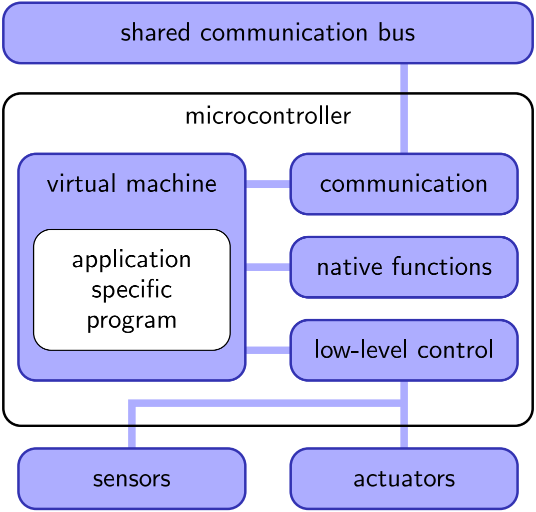
Aseba:

- allows fast prototyping of the behaviour of microcontrollers connected through a network,
- provides an IDE for edition, debugging and inspection of the values of variables (including sensor and actuators),
- compiles scripts into bytecode,
- executes bytecode on microcontrollers in a virtual machine,
- safe execution,
- dynamic enumeration of microcontrollers' variables, native functions and events,
- dynamic reprogramming of the microcontrollers,
- asynchronous code execution upon events,
- open source, LGPL.

Software Architecture



Microcontrollers



Virtual Machine

- targets 16-bit microcontrollers and better,
- stack based, 16-bit integers,
- executes bytecode
  (4-bit opcode, 12-bit payload + optional trailing 16-bit words)
- $\approx$ 1000 lines of C, including debugging logic,
- RAM: 22 bytes + user defined amount of bytecode, variable, stack, and breakpoints.
- flash: 7.5 kB flash (dsPIC30, e-puck),
- no external library requirement, excepted the implementation of bus communication.

Language

Simple imperative scripting language, octave-like syntax.

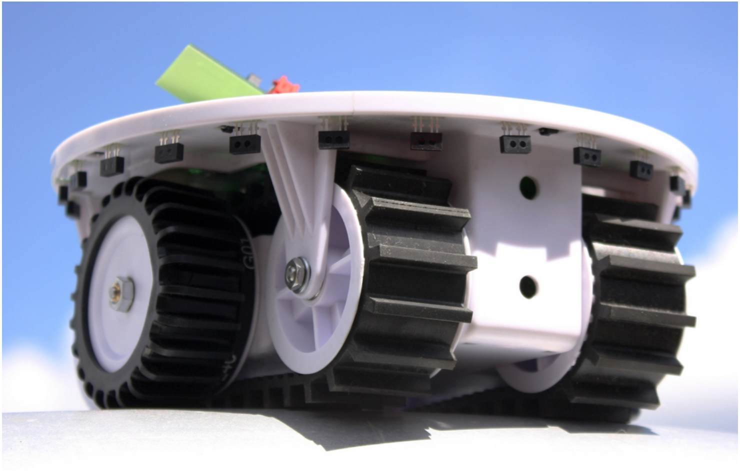
- blocks of code executed upon events,
- 16-bit integer variables and arrays,
- common mathematical expressions and arrays access,
- `if` and `when` conditionals,
- `while` and `for` loops,
- native functions for complex processing,
- subroutines.

Studio (IDE)

Live Demonstration using Thymio II

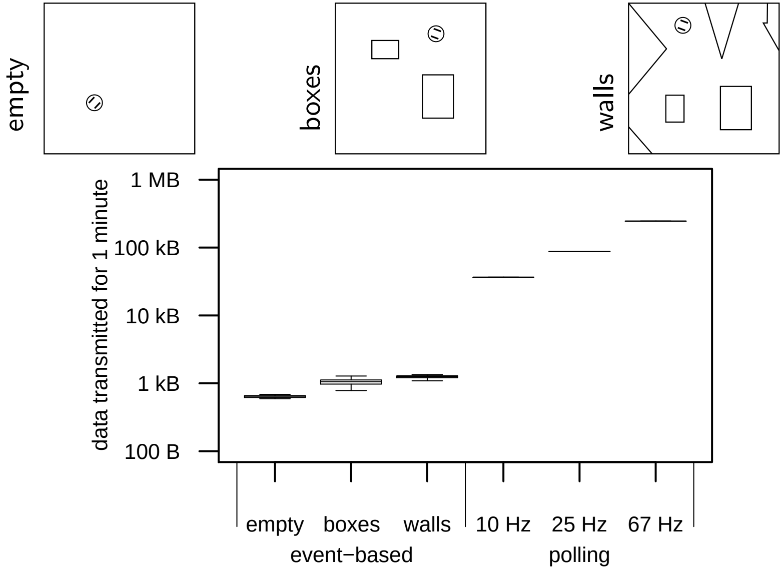
Outline

Evaluation Platform

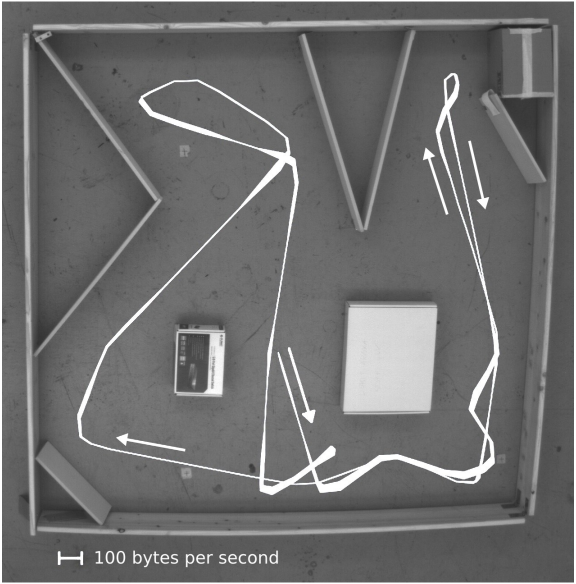


Evaluation conducted using the marXbot base.

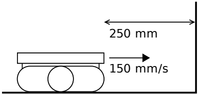
Aseba vs Polling: Bus Bandwidth



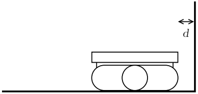
Aseba vs Polling: Bandwidth Use w.r.t. Location



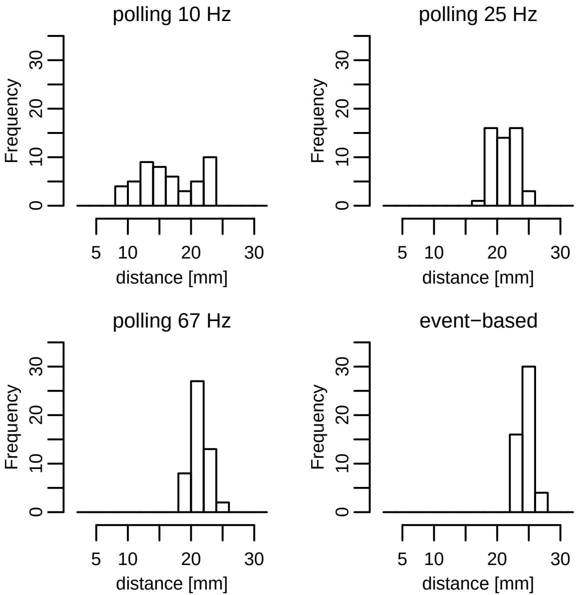
Aseba vs Polling: Latency



initially



once stopped



Aseba vs Native

On a 40 MHz dsPIC:

- Ratio of about 70 dsPIC instructions for 1 Aseba instruction
- Rate of 600'000 instructions per second
- Event round trip lasts 25 μs on idle VM
- Native functions in DSP assembly
- Native functions use dsPIC's DSP, event round trip for mean of 100 values is 60 μs, faster than bare C code

Lessons Learnt on Performance

- Events save bandwidth compared to polling.
- Events allow for lower latencies than polling.
- Virtual machines are suitable for embedded, provided the critical path is optimised.

Outline

Future Directions

Improve IDE

- store program as abstract syntax tree, textual surface form for editor
- contextualised help/errors
- intelligent completion
- statistics and machine learning for common mistakes, automatic tutoring

Lower entry curve

- improve tutorial
- robotics course
- more translations

More platform support

- new firmware for Thymio II
- affordable service robot based on Smartrob
- add your favourite platform!

On the long run

- no more arithmetic/logic operation in the VM, use only native functions
- complete type system
- more proving in the compiler
- JIT on some platforms
- use standard language?

Take-Home Message

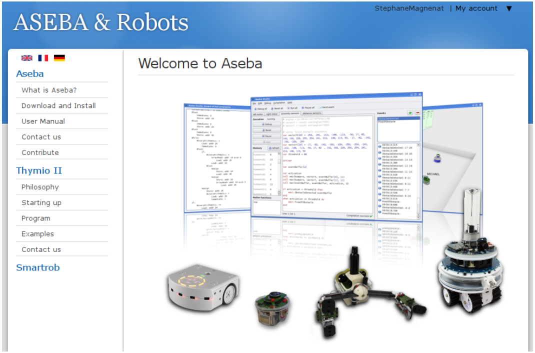
The combination of VM and custom-tailored IDE enables efficient embedded development, in particular for robotic applications.

Aseba is a mature and robust implementation of this idea. Yet there is much room for improvement and innovative ideas.

You are welcome to join us in this endeavour!

Thank you

Thank you for your attention, your questions are welcome.



The Aseba community awaits you: `http://aseba.wikidot.com`

Thanks to: Michael Bonani, Florian Vaussard, Fanny Riedo, Valentin Longchamp, Basilio Noris, Sandra Moser and Francesco Mondada

Program Memory Layout

addresses (in 16-bit words)	content
`bytecodeSize` −1 ...	unused bytecode
... `evLastAddr`	bytecode for last managed event
	...
... `ev0Addr`	bytecode for first managed event
`evVectSize` −1	`evLastAddr`
`evVectSize` −2	`evLastId`
...	...
`0x0002`	`ev0Addr`
`0x0001`	`ev0Id`
`0x0000`	`evVectSize`

Data Memory Layout

addresses (in 16-bit words)	content
`variablesSize` -1 `...`	temporary variables to pass constants to native calls
`...`	unused variables
`...` `exportedVarsLength`	user-defined variables
`...` `0x0000`	exported variables

Types of Bytecodes - 1/2

name	w.c.	function
stop	1	stop execution
small immediate	1	push a constant onto the stack
large immediate	2	push a constant onto the stack
load	1	push data from memory onto the stack
store	1	pop data from the stack into the memory
load indirect	2	push data from memory onto the stack using an offset from the stack
store indirect	2	pop data from the stack into the memory using an offset from the stack
unary arithmetic	1	unary arithmetic operation on the stack
binary arithmetic	1	binary arithmetic operation on the stack

Types of Bytecodes - 2/2

name	w.c.	function
jump	1	jump to another execution address
conditional branch	2	check a condition on the stack and jump depending on the result
emit	3	send an event
native call	1	call a native function
sub call	1	jump into a subroutine, store return address on the stack
sub ret	1	return from a subroutine, using return address from the stack