Roboscoop!

Project presentation
Spiez, 13 March 2013

Partners

ETH Zurich, Chair of Software Engineering

Hochschule Luzern, iHome Lab

ETH Zurich, Autonomous Systems Lab
Roboscoop project members

- Bertrand Meyer
- Benjamin Morandi
- Sebastian Nanz
- Andrey Rusakov
- Jiwon Shin
- Alexander Klapproth
- Dieter von Arx
- Martin Biallas
- Rolf Kistler
- Marcel Mathis
- Andreas Rumsch

Goals

Develop framework for concurrent robotics programming

Produce prototype of SmartWalker for elderly people

Learn about software engineering for robotics
Achievements so far

First version of Roboscoop framework

First version of SmartWalker

Setting up a course to teach this stuff!

Software architecture

**Roboscoop**

Integration with other frameworks, external calls

**SCOOP**

O-O structure, coordination, concurrency (wheels, navigation system, GUI, etc.)

**ROS**

Serial communication with hardware
Coordinate systems
Image processing
Navigation
...

...
**SCOOP**

Simple Concurrent Object-Oriented Programming

- Easy parallelization
- One more keyword in Eiffel (*separate*)
- Coordination is easy to express: close correspondence with behavioral specification
- Natural addition to O-O framework
- Retains natural modes of reasoning about programs

**SmartWalker**

Smart assistant for elderly people

Hi-tech extension of the regular walker

Autonomous robot with sensors and actuators

Possible functionalities:

- Support while going uphill/downhill
- Navigation during shopping
- Finding a charging station
- Fall detection
- ...

SmartWalker [image of a hi-tech walker]

[Image of elderly people using the SmartWalker]
SmartWalker

**Tablet PC running SCOOP**

**Single-board computer**

**Battery**

**E-bike motor controller**

**Powerful motors**

---

**SmartWalker: Hardware**

**Single-board computer (BeagleBone)**

- Low-cost
- Credit-card-sized
- 720MHz ARM processor
- Operating system: Linux computer
- Connectivity: 2 x 46 pin headers
More about the hardware

Motor controller: controls motors up to 350 W
- Controls force of motors according to given voltage & direction signal
- Optimized for controlling e-bike motors

Motor:
- Integrated Hall sensor to determine position
- Accumulator
- 36 V / 12 Ah with electronic management

SmartWalker: Software

Single-board computer
- Measures 2D position
- Speed control loop for each wheel
- Controls wheels concurrently

Tablet PC:
- User interface
- High-level control
- Roboscoop

Communication over ROS
Roboscoop

Coordination layer above SCOOP (and hence Eiffel)

Three-layer architecture (Gat, 1997):
- Deliberation
- Sequence
- Control

Synchronization: wait conditions

Interoperability through ROS (external calls)

Object & processor architecture
**SCOOP: separate calls** ("embarrassingly parallel")

walker_signaler: separate WALKER_SIGNALER
-- From sensors: position, orientation...

stop_signaler: separate STOP_SIGNALER

path_signaler: separate PATH_SIGNALER

actions: LIST [separate ACTION]

start_path (left, right: separate WHEEL)
-- Perform sequence given by actions.

local
  i: INTEGER
  do
    across actions a until
      stop_requested (stop_signaler)
  loop
    execute (a, path_signaler, stop_signaler, left, right)
    wait (a, path_signaler, stop_signaler, left, right)
  end
end

**SCOOP: synchronization through preconditions**

to_next (a: ACTION
  left, right: separate WHEEL
  ps: separate PATH_SIGNALER
  ss: separate STOP_SIGNALER
  ws: separate WALKER_SIGNALER)
-- Unless stop requested, complete a and enable next action.

require
  ss.stop_requested or (ps.state = a.index and a.done (ws))
  do
    left.stop
    right.stop
    if not ss.stop_requested then
      ps.set_state (a.index + 1)
  end
end
**SCOOP: wait by necessity**

```plaintext
path: LIST [separate ACTION]
executor: EXECUTOR
    -- To obtain actions from a script:
    path := parser.item (script)

    -- To execute sequence of actions:
across path as p loop add_action (executor, p.item) end

add_action (e: separate EXECUTOR; a: separate ACTION)
    -- Add a to action executor:
    local
    s: BOOLEAN
    do
    e.add_action (a)
    s := a.done -- The order matters!
    end
```

---

**Application to teaching**

New interdisciplinary course at ETH, Fall 2013:

**Robot Programming Laboratory**

Open to CS and ME students

**Combines software engineering, concurrency & robotics:**

- How software engineering applies to robotics
- Main architectures, coordination, synchronization methods
- Experience in programming a small robotic system
- Sensing, planning and control
- ROS and Roboscoop
Things we learned

Concurrency is great for robotics
A SmartWalker would be truly useful

This stuff is tough

Roboscoop: what’s next?

Gain more experience through course
Continue enhancing the Roboscoop concurrency framework
**Add sensors** to SmartWalker!
Implement SmartWalker scenarios
Evaluate applicability to other robots
Perform evaluation of SCOOP for robotics