Teaching Erlang using Robotics and Player/Stage

ACM SIGPLAN Erlang Workshop 2009

Sten Grüner and Thomas Lorentsen

RWTH Aachen / University Of Kent

5 September 2009

The Problem

Computer Science is losing popularity

- The number of CS undergraduates is decreasing
- Some CS departments were closed in UK

The Problem

Reasons for the bad image

- no connection to the real world
- "hacking for hacking's sake"
- pointless code debugging

How can young people be inspired?

- give lectures on a real-life context
- use cutting-edge libraries and tools
- let students contribute to open-source projects
- no "correct answer paradigms"

Our idea

Robotics fits perfect in this teaching concept

- interesting real-life topic with real-time problems
- robots fascinate people
- multiple solutions for a task possible

Erlang accomplishes robotics nicely

- robots are inherently concurrent
- descriptive language enforces conceptual thinking instead of solving hardware-related problems
- easy ad-hoc solving of concurrency problems

Our idea

KERL

Kent Erlang Robotics Library

- practical way of teaching Erlang
- simple API
  - emphasise on learning Erlang than learning KERL
- layered
  - build upon existing KERL functions
  - solve problems without rewriting code

Our idea

Real robots are rare in CS education

1. expensive to deploy and maintain
2. different vendors with incompatible, proprietary APIs

KERL solves these problems in terms of

1. using an open-source middleware
2. using simulation instead of real robots (the usage of real robots is, of course, still possible)

Our idea

Player

- open-source robotic middleware
- hardware-independent API for various vendors
- platform-independent, since driven via TCP
- wide-spread and used worldwide by labs and universities

Stage

- extends Player by providing an indoor 2D simulation
- the simulation is very reliable and physically realistic

Figure: A Pioneer 3-DX with a laser sensor and a video camera

Our idea

Player

- open-source robotic middleware
- hardware-independent API for various vendors
- platform-independent, since driven via TCP
- wide-spread and used worldwide by labs and universities

Stage

- extends Player by providing an indoor 2D simulation
- the simulation is very reliable and physically realistic

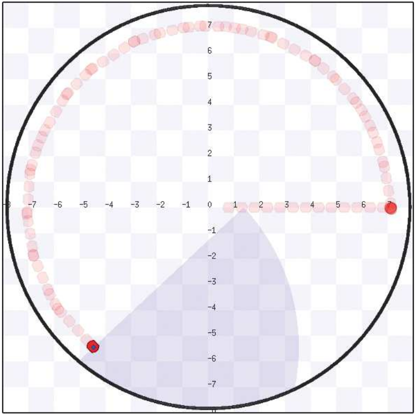


Figure: Stage simulating a Pioneer 3-DX

Architecture overview

User-level modules

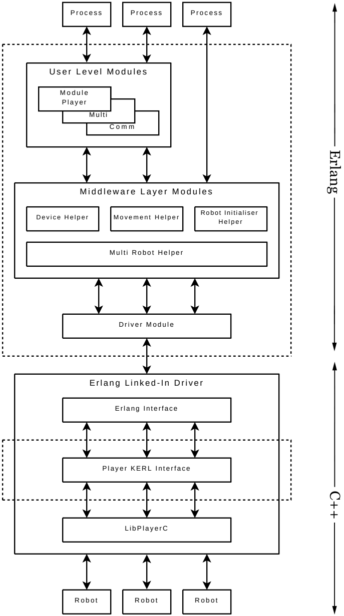
- easy learning of KERL
- comfortable concurrency management

Middleware

- robot initialization and control
- wrapping functions of Player

Linked-in driver

- manages the asynchronous LibPlayerC-Erlang communication



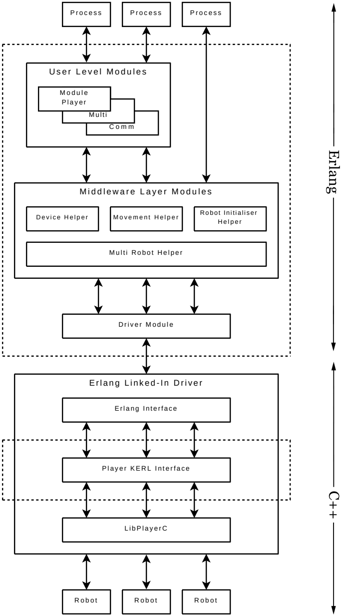
The linked-in driver

LibPlayerC utilised

- TCP protocol not documented
- LibPlayerC is documented
- the library is widely used

Linked-in driver

- Erlang-C communication
- asynchronous mode
- connecting LibPlayerC and EI
- hardest piece of work



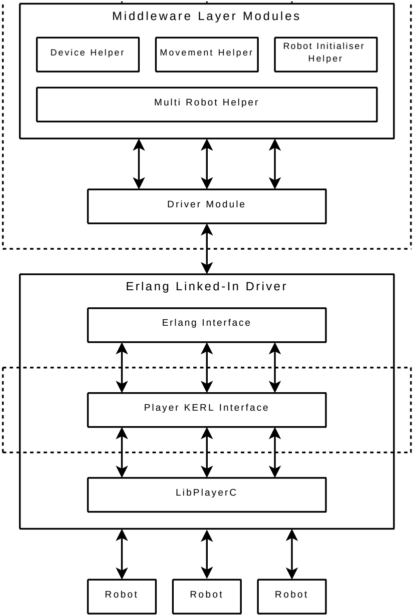
The linked-in driver

LibPlayerC utilised

- TCP protocol not documented
- LibPlayerC is documented
- the library is widely used

Linked-in driver

- Erlang-C communication
- asynchronous mode
- connecting LibPlayerC and EI
- hardest piece of work



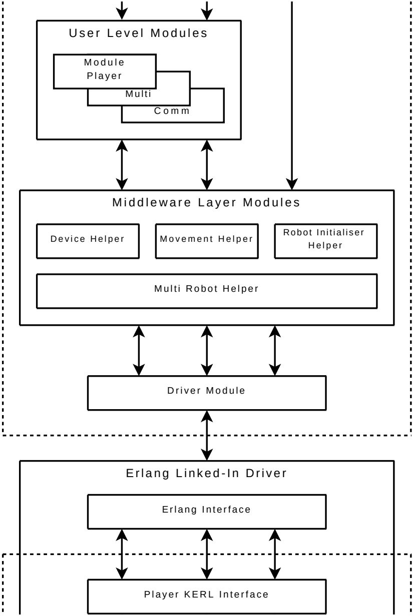
The linked-in driver

Driver Module

- initiates the linked-in driver
- passes messages to the linked-in driver
- receives messages from the linked-in driver
  - returns the results back to the caller

Multi Robot Helper Module (mrh)

- robot id is bound to a pid
- each robot is handled in its own process

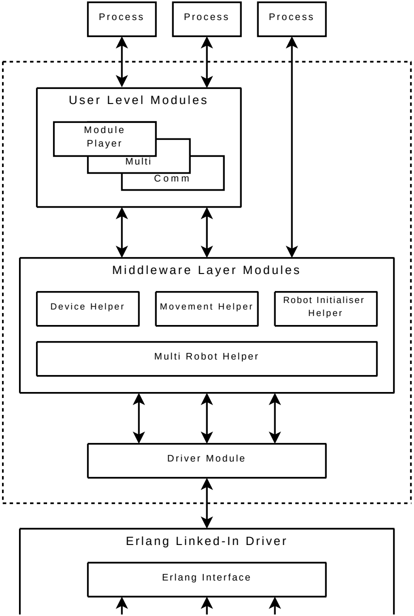


Middleware

Middleware Modules

- provide an API for
  - initalisation
  - movement
  - reading sensors
- simply structured
- non-blocking
- all modules provide fast and concurrent functions
- control multiple robots from a single function call

The middleware allows to write basic robotic applications.



Live examples of KERL usage

Driver initialisation

- use mrh module to start the KERL driver

```
1> Driver = mrh:start().
<0.34.0>
```

- multiple robots initialised from single driver instance
- keeps track of multiple robot instances

Live examples of KERL usage (continued)

Connect to a robot

```
2> Robot = rih:init(Driver, 0).
<0.37.0>
```

- controlled by passing the PID to many of KERL's modules

Connect to multiple robots

```
3> Robots = rih:plinit(Driver, [1,2,3],
                        [{host, "localhost"}]).
[<0.47.0>,<0.46.0>,<0.44.0>]
```

- done concurrently
- simplifies multiple robot initialisation

Live examples of KERL usage (continued)

Movement

```
4> mvh:move(Robot, distance, 1).
ok
```

- supports: distance, speed, position, difference

Rotation

```
5> mvh:rotate(Robot, degrees, 180).
ok
```

- supports: speed, degrees

Odometer

```
6> mvh:get_position(Robot).
{1.0,0,3.14159}
```

KERL in action

Laser sensors

```
1> trimaths:rad2deg(dvh:read_lasers(Robot)).
{[-89.99998127603168,-89.5013361069293,
  -89.00274823360647,-88.5041030645041 1,
  -88.0055151911 8127, -87.5069273178584 4,
  -87.00828214875607,-86.50969427543322,
  -86.01104910633087,-85.5124612330080 4,
|...],
 [3.94,3.94005,3.940456,8.0,8.0,8.0,8.0,8.0,8.0,8.0,
  8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0|...]}
```

- returns a pair of lists
  1. a list of bearings, here in 0.5° steps
  2. a list of distances (8m is the maximum)
- use this to sense obstacles like walls
  - implementing collision avoidance

KERL in action

Collision avoidance in Traffic Control Simulation

- robots navigate around a map with crossroads
  - avoiding walls and deciding which way to turn
  - read lasers every 1 meter

```
case lists:min(lists:zip(Distances, Bearings)) of
    {Distance, Bearing} when Distance < 1.5 ->
        case Bearing > 0 of
         true -> turn_right();
            _ -> turn_left()
        end;
    _ -> skip
end.
```

KERL in action

Use fiducial sensor to locate other robots and beacons

```
2>dvh:read_fiducial(Robot).
[{10,
  {1.038525,-0.588442,0.0},
  {0.0,0.0,-1.800621},
  {0.0,0.0,0.0},
  {0.0,0.0,0.0}},
 ...
 ]
```

Returns:

- list of found robots and beacons
  - beacon id (defined in Player world configuration)
  - relative Position (X, Y, Z)
  - rotation (Roll, Pitch, Yaw)
  - position and rotation uncertainty

KERL in action

Fiducial markers in the Traffic Control Simulation

- use fiducial sensor to detect nearby robots
  - queue behind robots
- pairs of beacons simulate a set of traffic lights
  - robot sees beacon ID with fiducial sensor
  - queries server for beacon state (Green/Red)

```
case nearest_beacon_state(dvh:read_fiducial(Rid)) of
    red -> stop();
    green -> move()
end.
```

- an easy task using KERL
- video available on youtube: http://www.youtube.com/watch?v=a93a1-uYyGk

User-Level Modules

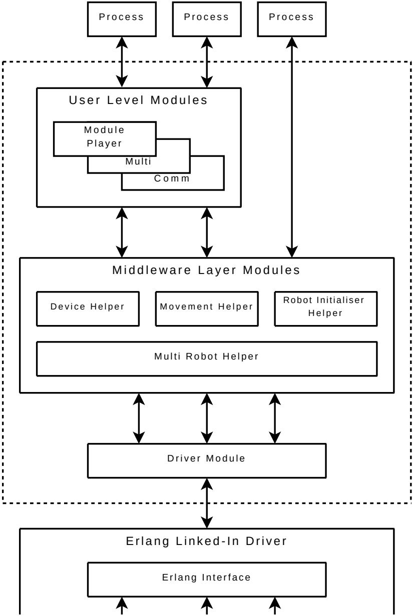
The modules bring some advanced functions into KERL

Player Module

- quick start with KERL
- simplified functions

Multi Module

- interprocess communication enhancements
- time-synchronisation, broadcasting, grouping of robots



User-Level Modules: Player Module

The module should be used for first student encounter with KERL

- all essential functions are concentrated in one module
- movement functions are artificially made blocking
- robot is bound to the process → only one robot per process
  - reduced number of arguments

player module vs. middleware

```
1> Driver = player:start().
<0.34.0>
2> player:init(Driver, 1).
<0.37.0>
3> player:move(distance, 1).
*blocks until stop* ok
4> player:get_position().
{0.983893,0.0,0.0}
```

```
1> Driver = mrh:start().
<0.34.0>
2> Robot = rih:init(Driver, 1).
<0.37.0>
3> mvh:move(Robot, distance, 1).
ok
4> mvh:get_position(Robot).
{0.983893,0.0,0.0}
```

User-Level Modules: Multi Module

The Multi Module provides inter-robot communication

- a group is implemented through a dispatcher process, which maintains a list of members and provides services
- robots can be included into a group of a dispatcher
- dispatchers can be members of a group as well

Functions available for group members:

- broadcasting messages to group members
- time-synchronisation between group members
- "voting" to select a unique leader

Multi Module: Usage

Creating a new group

```
1> Driver = mrh:start().
<0.34.0>
2> Dispatcher = multi:start().
<0.37.0>
```

Adding a process to a group

```
3> Process = spawn(?MODULE, main, [Dispatcher, Driver]).
<0.39.0>
4> multi:add(Dispatcher, Process).
ok
```

Multi Module: Usage (continued)

Time-Synchrnoisation

```
5> multi:barrier(Dispatcher).
ok
```

- called by a process
- unblocks as soon as every group member has called the function

Broadcasting

```
5> multi:broadcast(Dispatcher, {self(), {message}}).
ok
```

- called by a process
- sends to all group members except the sender

Multi Module: Usage (continued)

Voting

```
5> multi:vote(Dispatcher, true).
<0.39.0>
```

- called by a process
- flag indicates the participation
- blocks until all group members voted
- after everyone voted the winner is randomly chosen among participants

Multi Module in action

Multibouncer example

- robots move in a formation
- barriers are used to start moving synchronously (more or less)
- a stop signal is broadcasted as soon as one robot senses a wall
  - this robot becomes a leader
  - in case of more than one candidate the leader is selected by voting
- the remaining robots follow the instructions of the leader
  - the difference between the initial and the final position of the leader is broadcasted in order to realign the robots

Video available on youtube:
http://www.youtube.com/watch?v=39r207hFE6A

Infrastructure

One of the project aims is to provide an out-of-a-box teaching environment for Erlang courses. Hence, we provide a rich infrastructure with KERL:

Scripts

An automated installation of Player/Stage (more details later).

Examples

Basic KERL usage scenarios:

- bouncer
- wall follower
- group action examples (as just seen)
- fiducial sensor based spatial synchronisation (from the beginning)

Infrastructure (continued)

Tutorials

The tricky examples are explained in a walk through manner, e.g. the usage of multi group voting facilities and broadcasting.

Assignment ideas

The typical assignment is to extend an existing example, e.g. make the wall follower to be able to follow non-convex walls. We plan to add more assignment ideas to KERL.

Live CD

A modified Ubuntu Live-CD is available for download. The CD has Player/Stage and KERL preinstalled and can be used for testing KERL, as well as for comfortable installations.
The ISO image can also be run comfortably in VMware.

Getting KERL

KERL is available under GPL from http://kerl.sourceforge.net

Installing KERL is easy

- install guides available
- script speeds up Player/Stage and Kerl installation on Ubuntu
- installation simple on Fedora

KERL is portable

- KERL runs on any system that supports Erlang
- Stage was tested on Fedora, Ubuntu and OSX
- Windows support only via VMware (Stage limitations)
  - Stage performance limitations because of the lack of hardware OpenGL support

The future of KERL

KERL is easy to extend

- modular structure
- well documented
- let students contribute!

Planned improvements

- more devices to be added
- implementing the Player TCP protocol in Erlang
- examples, tutorials, assignment ideas
- Player 3 support
- Simplifying the installation
  - 1 click install
  - RPM

Summary

If you do not remember anything about KERL – remember this:

- KERL is a library which connects the Player/Stage robot simulator with Erlang
- KERL provides an out-of-a-box teaching environment for an Erlang course
- KERL is easily teachable and follows Erlang's philosophy
- KERL helps to inspire students and brings Erlang into the educational domain

Summary

If you do not remember anything about KERL – remember this:

- KERL is a library which connects the Player/Stage robot simulator with Erlang
- KERL provides an out-of-a-box teaching environment for an Erlang course
- KERL is easily teachable and follows Erlang's philosophy
- KERL helps to inspire students and brings Erlang into the educational domain

Any questions?