PLAYER AND STAGE AT PJIIT ROBOTICS LABORATORY

Received 4th July; accepted 16th Julyl.

Paweł Ośmiałowski

Abstract:

Player/Stage/Gazebo is an Open-Source Software project designed for robotics research that provides infrastructure for distributed access to robotics equipment both real and simulated. Founded by Brian Gerkey, Richard Vaughan, Andrew Howard, Kasper Stoy and Nate Koenig it soon became popular among roboticists community. Project itself gained lots of contributors and many related software projects were started that support or make use of Player/Stage/Gazebo software. At PJIIT Robotics Laboratory we decided to deploy this software for educational purposes as well as for our Virtual Robotics Laboratory project where it plays significant role as an integration platform. Since it is Open-Source Software we were able to contribute new modules and also release fixes for bugs that we have found. This paper deals with our experiences with using and maintaining Player and Stage software.

Keywords: *Robotics software, simulation software, open-source software, robotics programming, robotics laboratory*

1. Introduction

Player/Stage/Gazebo[1] is an open-source Software project that consists of three main software parts that make it successful robotics framework. It can be downloaded for free from projects home page (http://playerstage.sourceforge.net). These parts are: **Player** - the core of whole framework, which contains message-passing server and drivers for most popular robotics equipment; it can be extended by plugins - pieces of software that use Player as communication interface. **Stage** - a plugin for Player that acts as a 2D simulation device, which simulates existence of real hardware in predefined 2D workspaces, called worlds. **Gazebo** - a simulation plugin for Player that works with 3D workspaces. All this suite of programs and com panion software conforms the spirit of UNIX operating system [2] and Open-Source movement [3]. In this paper we will focus on Player and Stage parts only, as only these parts were used in PJIIT student's projects so far. Examples of these projects will be also described shortly.

2. Architecture

Whole framework makes use of client-server architecture where Player itself plays the role of the server while client programs communicate with it to access data (sensors readings, camera images, odometry or GPS position clues and so on) and to send commands to actuators (robot motors, gripper, pan-tilt-zoom module on camera and so on). There are programming libraries available for many popular languages (C, C++, Java, Lisp, Scheme, Python, Ada, Octave/Matlab) that are required to build client-side programs. Also Player server plugins may act as the clients for other Player servers, which is useful for building more sophisticated communication topologies.

Player may be considered as a communication bus, to which client-side programs can connect in order to communicate with robotics hardware through drivers also connected to that bus. Server configuration file describes what drivers are connected to communication bus in certain Player server instance. Although many drivers are *built-in* to Player itself, new drivers for the new hardware can be written in C++ language using Player API. These drivers are called plugins. Stage is an example of the most sophisticated Player plugin that provides access to simulated hardware.

Each driver provides at least one predefined interface (consequently, Stage simulator provides many different interfaces). Every interface describes the kind of offered data with their internal structure and syntax of commands that associated device can accept (except read-only devices that do not accept any commands, only provide data like camera images, sonar readings, laser scans, power source status, etc.). For example, *Video-4Linux* driver provides *camera* interface, which accepts no commands while data offered by this interface are: image (as an matrix of integer numbers), compression information (if JPEG compression is used or not), width, height and colour depth of the image.

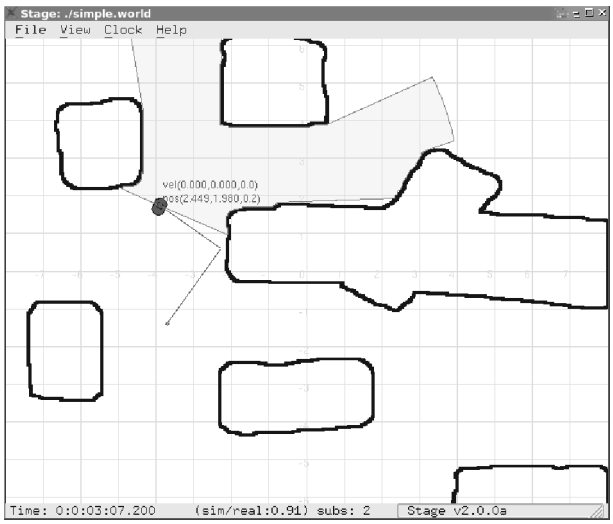

Fig. 1. Stage is the most sophisticated plugin for Player - it opens its own window in which progress in simulation can be observed; also a user can make changes in simulation by moving objects using a mouse.

There are some drivers (both built-in and plugins) that aren't associated with any particular hardware. These drivers provide useful functionality for whole Player infrastructure. Since communication between drivers is also possible through the Player's communication bus, one driver can take data in one form and offer different representation of them for connected clients or other drivers. Also one driver can take data directly from other Player server (it acts as a Client then) and then it can offer data processed by itself to other drivers or clients connected to Player server in which this driver was started. Note that communication between connected clients using Player communication bus is not possible; they should use other means of communication if they need it.

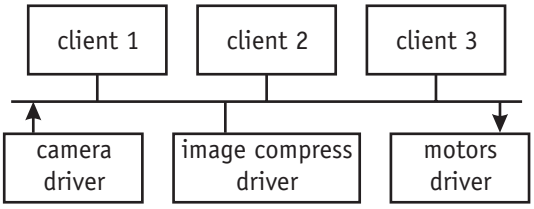

Fig. 2. Player communication bus with connected clients and drivers.

As an example of a driver that is not associated with any certain hardware we may consider cameracompress that takes image from camera driver and offers the same image JPEG compressed, which can be useful for distant communication purposes. Another example is the *amcl* driver that implements Adaptive Monte-Carlo Localization algorithm. It takes data from other drivers using *position2d* interface which offers odometry position clues, laser interface which offers *laser* scans of environment nearby, *map* interface which offers 2D map of the whole current workspace and provides *position2d* interface to return guessed robot position within given map. In facts, *position2d* interface has two roles: to provide current position (absolute or odometric) and to command position changes which effects in robot motors activity.

There are two kinds of position change commands accepted by *position2d* interface: position commands (for example *move to given distance from current position*) and velocity commands (for example *move forward with given speed*). Which kind of commands is accepted depends on associated device capabilities. Since real robots accept only velocity commands, a driver called *vfh* (Vector Field Histogram) can be used, which takes position commands using *position2d* interface and recalculates them to velocity commands that are later sent to *position2d* interface provided by some robot's driver. This driver also uses *laser* interface to read laser scans necessary to obstacle avoidance during the movement. Note that metric system of units is used in Player for distance measurements.

There are three kinds of data travelling through communication bus: sensor readings, commands and configuration queries. First two were explained above. Configuration queries are sent to certain driver, which responds to it instantly. They are accepted both by read-write and read-only devices. For example, configuration query can be sent using *laser* interface to ask for geometry of laser device (where it is situated on the robot and what is it's shape and size, information typically used by client-side graphic visualization programs that tries to restore current situation in users application window). The same query can be sent using *position2d* interface to ask for - geometry of whole movable robot. Also configuration query can be sent to change some parameter of the device (even if it's read-only). For example, laser scans resolution can be changed that way.

Some of the drivers that are not directly assigned to hardware can do heavy computations. If Player is intended to run on robot's onboard computer it must be considered that it can be too slow to run some of mentioned drivers that Player can provide. For example, there are few drivers that do advanced image processing. They are linked against **OpenCV** [17] (Open-Source Computer Vision library), which provides programming functions for image processing that needs CPU-consuming computations. Mentioned earlier *amcl* driver (Adaptive Monte-Carlo Localization) needs GNU Scientific Library [18] (**gsl**), which also provides functions that do heavy computations.

3. Client-side programs

Player software package is shipped with example client-side applications. Although desired way of using robotics devices with Player is to write a program implementing control behaviours that does not require interaction with human operator, first program that most users learn first is **playerv** (PlayerViewer) - graphics user interface for controlling interactively all the devices accessible by Player server to which it will be connected. To control a device or to just read data from it a subscription should be made using *Devices* menu. The way subscribed device is represented on the window work area depends on interface provided by its driver. See figure 3 for an example situation.

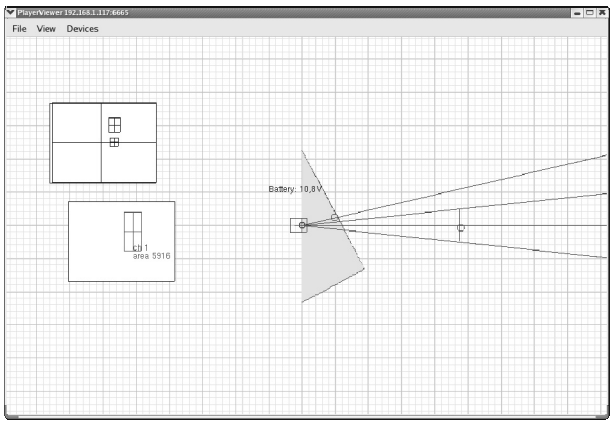

*Fig. 3. PlayerViewer (**playerv**) connected to a Player server that works on Pioneer P2DX robot on-board computer. As we can see, there are several devices subscribed that provide following interfaces: position2d (red box), ptz (blue and green lines, pan-tilt-zoom driver for the Sony EVID-30 camera), laser (blue field represents free space seen in front of the robot) and two different devices that provide blobfinder interface.*

Unfortunately, **playerv** cannot manage with camera interface; therefore another client program called **videoplayer** [15] was created which is an example of many contributed applications. It opens a window in which image from camera interface is constantly redrawn (see figure 12). It is also able to check if an image is JPEG compressed and run decompress routine to manage with it. Later Player developers have made another program that does similar thing called **playercam**, but **videoplayer** is still in use due to its simplicity and clear source code which is used as an example of how camera interface data can be acquired in programs for further image processing.

Another interesting client-side program shipped with Player software package is **playernav**. It is designed to be a *frontend* for the *planner* interface (yet another interface that **playerv** cannot manage with). It displays map of the current workspace taken from given device that provides *map* interface. A user first has to place (using mouse) robot symbol within this map to give **playernav** a hint where the robot currently is and what is its current pose. Then it is possible to set the target position, which will be sent to the device that provides *planner* interface.

Example driver that provides such interface is *wavefront* - a path-planning device, which, after receiving new target position a robot should approach, computes list of waypoint positions, that makes a patch to the target position. During these computations, *wavefront* helps itself by using a map of the workspace taken from a device that provides *map* interface. Having this list of waypoint positions, *wavefront* sends commands to actuator device using *position2d* interface. Since majority of known *position2d* actuator devices accept only velocity commands, typically *wavefront* sends position commands to the *vfw* device which changes them to desired velocity commands as described earlier. Commonly used device that provides *map* interface is called *mapfile* - it reads bitmap image from *.pgm* or *.png* file and interprets it as a workspace map with given resolution (for example 10cm per pixel). Each black pixel denotes place occupied by an obstacle, while white pixels denote free space.

4. Player and Stage at PJIIT

Player and Stage are widely used in PJIIT Robotics Laboratory. It is used both for educational purposes (during classes) and for research projects. The success of this robotics framework corresponds to overall success of Open-Source Software in field of educational research at PJIIT.

Fig. 4. ***Playernav*** *in action; black rectangle is the map of the workspace available in our Virtual Robotics Laboratory.*

Despite of that, the most important role at PJIIT Player infrastructure plays in Virtual Robotics Laboratory [16] where it integrates all communication.

4.1. Player as an integration platform for Virtual Robotics Laboratory

The purpose of Virtual Robotics Laboratory is to provide remote access through the Internet to its robotics resources for students and researchers from other educational institutes. At the beginning of the project, very first topic to consider was the choice of integrated communication platform [4]. Although we could choose to create our own communication software or to use existing solutions like CORBA, we wanted to conform standards of software used in robotics research laboratories around the world. We've realized that the most popular Open-Source Software solution is the Player server, which was already used by individual PJIIT students for their projects.

Currently our Virtual Laboratory offers access to two Pioneer P2DX robots (one of them is adapted to work 24 hours 7 days a week, the other is on duty at users request). Each is equipped with ring of ultrasonic sonars, SICK LMS200 laser, camera Sony EVID-30 with pan-tilt-zoom module, and onboard computer (PC/104+ compatible) with *Video4Linux*-compatible PCI framegrabber device. One of them is also equipped with gripper. To observe current situation, there are few cameras situated on top of the workspace. Two of them are mounted on movable tractor devices controlled by Player's plugin running on PC computer, which access these devices through RS232C serial port. This plugin driver was created in our laboratory and provides *position2d* interface making remote operator able to change position of given camera.

Fig. 5. One of D-link cameras mounted on a movable tractor.

Each PC computer and all onboard PC/104+ computers working in our Virtual Laboratory has started at least one Player server instance. The main server working at *address vlab.pjwstk.edu.pl* at TCP port 6665 offers public access to cameras and full access to movable tractor devices. Other devices (for example robot actuators) are available through special authorization proxy that we have designed to make sure only registered users who

have full responsibility on the way they use laboratory resources can have full access. These users can also access our Control Panel web service available at address https://cp.vlab.pjwstk.edu.pl. Users, that use this control panel, can view the history of their access attempts and current condition of every device provided by our Virtual Laboratory. Supervisor user can also disconnect all other users, which leads to stop all robots movement.

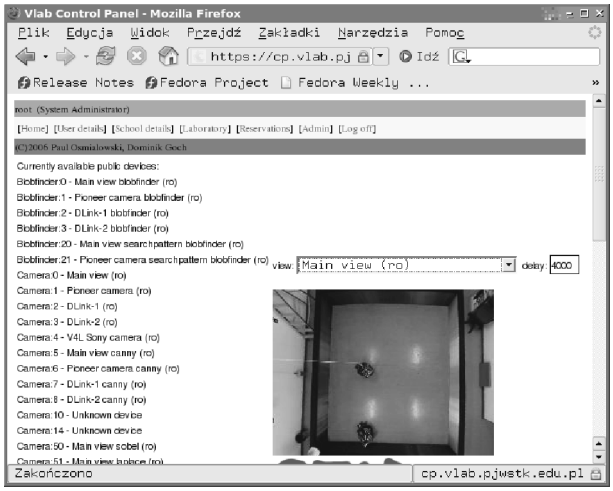

Fig. 6. Virtual Laboratory Control Panel started in the Firefox web browser.

Currently our laboratory uses Player version 1.6.5 with our set of bugfixes. Corresponding client software and programming function libraries for both UNIX-compatible and MS Windows systems are available to download. We're providing source code, binaries for MS Windows and a portage tree for automatic installation in **Gentoo Linux** operating system [20], which is the most supported system by us.

4.2 Student projects

Janusz Matkowski used the first time Stage simulator at PJIIT in his graduate studies research [5]. In his work he has described approach to artificial intelligence opposing classic AI ideas (this approach was earlier presented by Rodney A. Brooks [6]). Instead of heavy symbolic processing, paradigm of *embodied intelligence* relies on physical implementation (in particular environment) as a key to achieve truly intelligent agents. Aside from describing theory behind this approach he has demonstrated in simulation (figures 7 and 8) how sophisticated behaviours can emerge from appropriate combination of agent morphology and controller.

Another example of students' project is Karol Yamazaki's PaGo (*Point-and-Go*) [7]. The main goal was to create new means for mobile robot navigation based on image analysis. Yamazaki's program is using Player to acquire live image from robot's camera. Image analysis is used for pointer object detection and space positioning. This pointer object (yellow bar) role is to show a robot where it should go (see figure 9). **OpenCV** [17] library functions were used for image analysis.

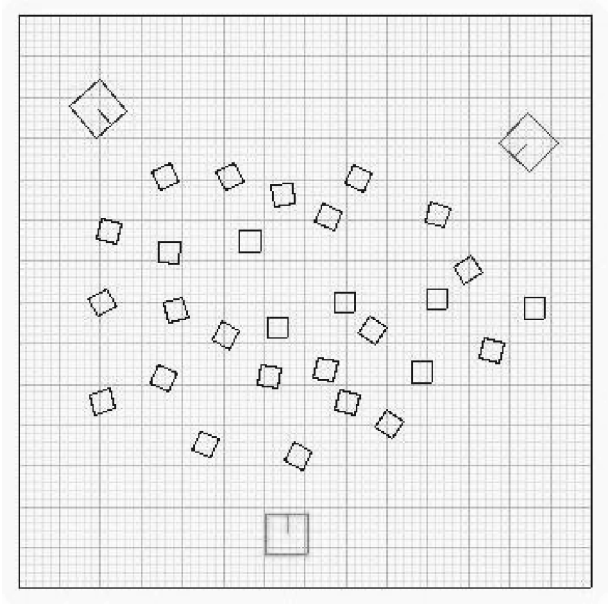

Fig. 7. Snapshot from Stage simulation of one of experiments described by Janusz Matkowski in his work [5].

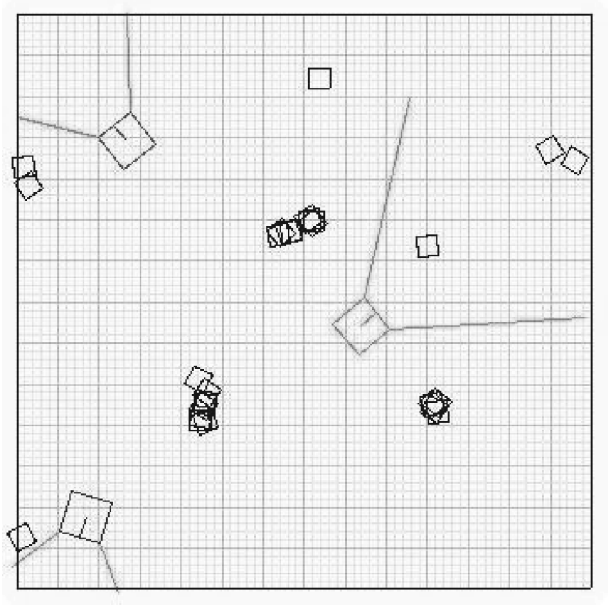

Fig. 8. Snapshot from experiment shown in previous figure after few minutes of simulation process.

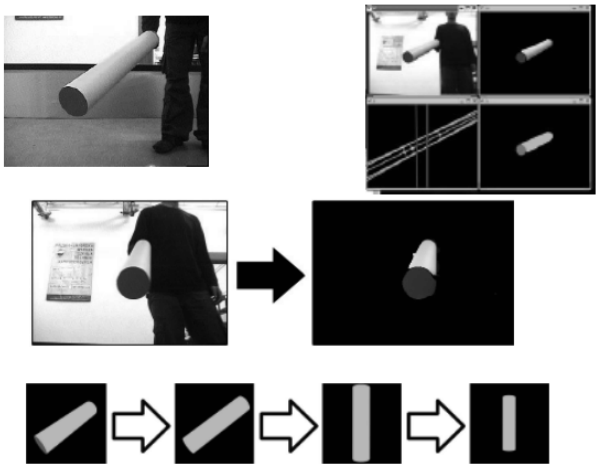

Fig. 9. PaGo at work.

Potential Field Method (PFM) as means for navigation is used in Michal Dendewicz and Lukasz Hrynakowski project [8]. The method is not a new idea, but is still considered interesting, so they wanted to focus on it and present its main assumptions on an operative model. They used Pioneer P2DX robot equipped with SICK LMS200 laser device and onboard computer running Player server.

In several words, PFM main assumption focuses on imaginary forces acting on a robot. It can be compared to an electron behaviour in an electromagnetic field. In their case, mobile robot is the electron, and the electromagnetic field is emitted by obstacles situated in the robot's workspace (figure 10). Although PFM is very effective and pretty simple, it may suffer from local optima problem. Our students in their work have described their challenges in avoiding it.

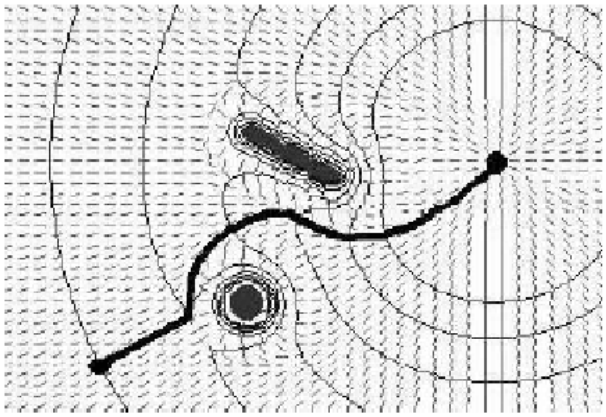

Fig. 10. Example potential field.

4.3. Our improvements and bugfixes

During the years of experience with Player and Stage software we have made number of improvements and bugfixes. First changes in Player's code were made in parts responsible for FireWire cameras operation. In our Robotics Laboratory we are using Imaging Source's DFK 41F02 high resolution FireWire camera that we wanted to use as a live image source for Player. Trying to do it we have realized that whole Player driver for FireWire camera devices should be totally rewritten.

After buying three D-link's DCS-5300W cameras we had to write a completely new Player driver for them. It provides two interfaces: *camera* for the live image and *ptz* for pan-tilt-zoom module available in these cameras.

We have also discovered and fixed few bugs in *Video4Linux* driver for cameras connected to the standard PCI framegrabbers.

Although Player developers try to be up to date with providing drivers for the most popular robots, there are still many devices that need new drivers to be written. Example is small Hemisson robot by K-Team. PJIIT Robotics Laboratory uses two such robots. There are Player drivers for other K-Team products built into Player server (Khepera robot, and robot controllers based on REB/Kameleon board). Hemisson uses communication protocol similar to other K-Team robots; therefore we were able to base our work on Khepera driver source code written by Toby Collett from University of Auckland Robotics Group. During this work, we did totally change serial port communication part of the driver code using new functions based on source code of **minicom** [19] program (popular Open-Source communication software).

During the time we were using Player, many changes has taken place in related Open-Source Software infrastructure. When the new version line (4.x) of **gcc** compiler suite that is used to compile Player was started, some smaller parts of the code had to be rewritten. Also new, more restrictive version of GNU C library (**glibc**) made some hidden errors in memory management to show up. We have proposed set of patches that fix both problems for Player version 1.6.5 and Stage 2.0.0a. Also we have released portage tree for **Gentoo Linux** [20] that provides building guidelines for Player 1.6.5, Stage 2.0.0a and Gazebo 0.5.2, all including our bugfixes.

For MS Windows users we have ported client-side programming library using **MinGW** [21] development environment. We had to replace usage of **BSD sockets** (used for TCP communication in Linux and other UNIX-compatible systems) with **Winsock**. Having client-side library ported to Windows we were able to release **playerv** and **videoplayer** compatible with Player 1.6.5 as regular Windows applications [22].

Soon we have realized that instead of using two different models of communication sockets, one for MS Windows, another for other systems, we can use one portable solution that works the same way almost everywhere. We have used **SDL** suite of highly portable programming libraries [23] that also provides its own communication sockets in a programming library called **SDL_net**. That way we have released SDL-style version of client-side programming library and using this we were able to release new **playerv** and **videoplayer** as regular Windows application, this time compatible with Player 2.0.4 [24].

During the works on Virtual Robotics Laboratory project we have released two simple programs that constantly monitor state of our infrastructure [9]. The **playercheck** program checks every 30 seconds if given Player server started on the same host responds properly (it tries to read list of available devices). If not, the process of that Player server instance is killed. Since we're starting our server instances in infinite loop, new Player server will be started in these circumstances. Another monitoring program is called **lowpower**. It is started on a robot which Player's driver provides *power* interface. It reads constantly voltage and whenever it goes below defined threshold, whole system is going to shutdown. This protects at least file system from being damaged during unexpected halt of onboard computer.

Although C, C++ and Java are of the most popular programming languages in PJIIT, students in our Robotics Laboratory use also other languages for their programming works. Recently, Octave [10] - Open-Source Software interpreter of Matlab-compatible programming language became popular, mostly during the classes. We have started to use Cameron Morland's **octplayer** [11] - Player's client-side programming library for Octave. Soon we have extended it by adding more interfaces (laser, camera, map, localization). Also we have rewritten it totally to become compatible with latest Player 2.0.4 (June 2007).

```
r.robot = client_create("vlab.pjwstk.edu.pl", 6665);
r.index = 1;
laser = proxy_create("laser", "r", r);
n = 0;
while (1)
  n++;
  do
    for i = 1:5,
      if (client_read(r))
        error("client_read returned an error!\n");
      end
    end
    scans = laser_val(laser); len = length(scans);
  until (l > 359)
  x = cos((pi / 180.0) * (((1:len) - 1.0) / 2.0)) .* scans(1:len);
  y = sin((pi / 180.0) * (((1:len) - 1.0) / 2.0)) .* scans(1:len);
  clg();
  plot(x, y, "@");
  if (n > 200)
    n = 0;
    closeplot(); purge_tmp_files();
  end
end
closeplot(); purge_tmp_files();
client_destroy(r);
```

Listing 1. Example Octave script that constantly plots scans from laser device.

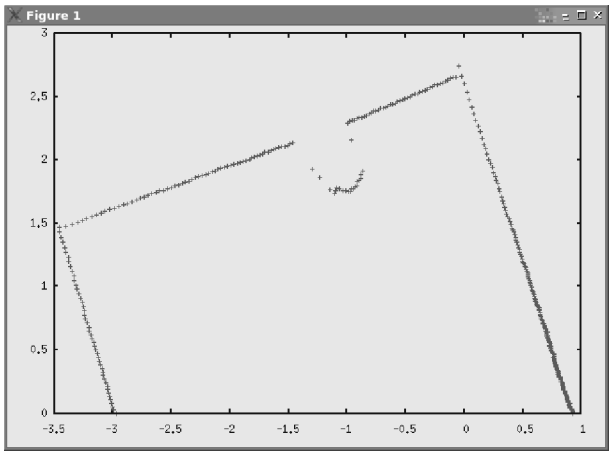

Fig. 11. Output of a script presented in listing 1.

Another programming language popular in our Robotics Laboratory is Scheme. We have developed client-side programming library called **guileplayer** [12], which is intended to use with **guile** [13] - the most popular Open-Source Software interpreter of Scheme. Writing this library we tried as hard as possible to conform the spirit of Scheme language, therefore we couldn't have used automatic bindings generators (like **swig** [14]). We had to do everything from scratch, but finally we can admit that it was worth it.

Fig. 12. The same situation as in figure 11 seen by the robot's camera (it cannot see 180 degrees as laser do, so this image shows less information about obstacles than laser scans presented earlier).

```
(load-from-path "guileplayer.scm")

(define main-loop (lambda (client position sonar turn-counter)
    (cond
        ((not (player-client-signalled?))
      (player-client-read-for-sure client)
      (main-loop client position sonar (cond
     ((> (player-sonar-scan-count sonar) 5)
         (cond
       ((< (player-sonar-scan-n sonar 2) 1.0)
           (player-position-set-cmd-vel
                          position
                          '(0.0 0.0 -1.0)
                          player-enable
                      )
           (+ turn-counter 1)
       )
       ((< (player-sonar-scan-n sonar 5) 1.0)
           (cond
         ((> turn-counter 10)
            (player-position-set-cmd-vel
                               position
                               '(0.0 0.0 -1.0)
                               player-enable
                           )
         )
         (else
            (player-position-set-cmd-vel
                               position
                               '(0.0 0.0 1.0)
                               player-enable
                           )
         )
          )
           (+ turn-counter 1)
       )
       (else
           (player-position-set-cmd-vel
                              position
                              `(,(player-sonar-scan-n sonar 3) 0.0 0.0)
                              player-enable
                          )
          0
       )
        )
     )
     (else
         (display (player-sonar-scan-count sonar)) (newline)
        turn-counter
     )
      ))
        )
        (else
            (display "going to quit...")
            (newline)
        )
    )
))

(define client (player-client-create player-null "localhost" 6665))
(player-client-connect client)
(define position (player-position-create client 0))
```

```
(player-position-subscribe position player-all-access-mode)
(define sonar (player-sonar-create client 0))
(player-sonar-subscribe sonar player-read-access-mode)
(player-position-enable position player-enable)
(player-client-trap-signal SIGINT)
(player-client-trap-signal SIGTERM)
(main-loop client position sonar 0)
(player-position-set-cmd-vel position '(0.0 0.0 0.0) player-enable)
(player-position-enable position player-disable)
(player-position-unsubscribe position)
(player-position-destroy position)
(player-sonar-unsubscribe sonar)
(player-sonar-destroy sonar)
(player-client-disconnect client)
(player-client-destroy client)
```

Listing 2. Example Scheme script that implements very simple obstacle avoidance behavior. It uses sonar ring to detect distance to the obstacles.

5. Future work

Currently we are preparing for upgrade of whole Virtual Laboratory infrastructure to be compatible with Player 2.0.x. Also we are intended to help in developing new version line (2.1) of Player, which is supposed to be more portable and able to run also natively also on MS Windows.

Since PJIIT Robotics Laboratory students build their own robots, there is unavoidable need for writing Player drivers for them. A driver will be also required by a new global positioning device that is currently under construction and will be installed in our Virtual Robotics Laboratory.

AUTHOR

Pawel Osmialowski - Polish-Japanese Institute of Information Technology, ul. Koszykowa 86, 02-008 Warszawa, Poland, e-mail: newchief@ai.pjwstk.edu.pl.

References

[1] Brian Gerkey, Richard T. Vaughan and Andrew Howard. "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems". In *Proceedings of the 11th International Conference on Advanced Robotics* (ICAR 2003), Coimbra, Portugal, June 2003, pages 317-323.

[2] Eric Steven Raymond. "*The Art of Unix Programming*", on-line book available at http://www.faqs.org/docs/artu, 2003.

[3] Sam Williams. "*Free as in Freedom: Richard Stallman's Cruscade for Free Software*", on-line book available at http://www.faifzilla.org, 2002.

[4] Paul Ośmiałowski. "*Implementation of distributed robotics framework and robotics hardware adaptation in the Virtual Robotics Laboratory*", PJIIT, Master's thesis, Warsaw, Poland, June 2006.

[5] Janusz Matkowski. "*Paradygmat Inteligencji Ucieleśnionej*" [Paradigm of Intelligence Materialized], PJIIT, Master's thesis, Warsaw, Poland, October 2003.

[6] Rodney A. Brooks. "*Elephants Don't Play Chess*", MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, 1990.

[7] Karol Yamazaki. "*PaGo. System nawigacji oparty na analizie orientacji 3D obiektu*" [PaGo. The navigation system based on orientation analysis of 3D object], PJIIT, Warsaw, Poland, 2006

[8] Michal Dendewicz, Lukasz Hrynakowski. "*Potential Field Method for Mobile Robot Navigation*", PJIIT, Warsaw, Poland, 2006

[9] http://vlab.pjwstk.edu.pl/files/PJIIT/vlabrelated

[10] http://www.gnu.org/software/octave

[11] http://cns.bu.edu/~cjmorlan/robotics/octplayer

[12] http://sourceforge.net/projects/guileplayer

[13] http://www.gnu.org/software/guile

[14] http://www.swig.org

[15] http://king.net.pl/playercontrib/videoplayer

[16] http://vlab.pjwstk.edu.pl

[17] http://sourceforge.net/projects/opencvlibrary

[18] http://www.gnu.org/software/gsl

[19] http://alioth.debian.org/projects/minicom

[20] http://www.gentoo.org

[21] http://www.mingw.org

[22] http://vlab.pjwstk.edu.pl/downloads

[23] http://www.libsdl.org

[24] http://vlab.pjwstk.edu.pl/files/PJIIT/SDL