Diving into Educational Robotics with Player/Stage

Carlos Jaramillo

Computer Engineering Department, City College of New York, CUNY
New York, NY 10031 USA
cjarami00@ccny.cuny.edu

***Abstract*— This paper describes my first research experience in programming robots by using open-source software tools such as Player/Stage. Among the educational robots available at Brooklyn College's Agents Lab, the Surveyor and the Scribbler were preferred. Eventually, the project focused itself on implementing a plug-in driver for the Surveyor robot so that it could be controlled via Player/Stage. A Player/Stage-based GUI console - to remotely control the surveyor and other robots with similar actuators and sensors – was also developed although for now it lacks some features that could allow Simultaneous Localization and Mapping (SLAM) for Urban Search and Rescue (USAR) missions.**

***Keywords*— Player/Stage, surveyor, SRV-1, Scribbler, driver, SRVjoy, educational robotics, open-source, USAR, Brooklyn College, Sklar**

I. Introduction

This document is a final report from my summer 2009 distributed research experience as an undergraduate student. My mentor, Elizabeth Sklar, Ph. D, from the Dept. of Computer Science at the City University of New York's Graduate Center, directed my effort in this project for adapting drivers for the small educational robots available at the lab into Player/Stage [1] – a robot server and simulation software – and eventually designing and implementing a graphical user interface (GUI) for a human operator to send commands and receive input from the robot(s) under control via the Player server.

The Agents Lab at Brooklyn College has at researchers' disposition a wide range of robots. Among the small robots – employed in some CIS courses such as Exploring Robotics (CC 30.03) [2] and also used during the Bridges High School Workshop [3] – are the LEGO's Mindstorms RCX Robotics Invention System [4], IPRE's Scribbler [5] and Surveyor Corporation's SRV-1 [6]. By trying these robots through their available tools, we decided to give priority to the Scribbler (Fig. 1) and the SRV-1 (Fig. 2) due to their open-source centric implementations.

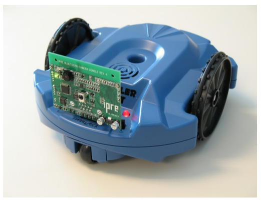
Fig. 1 Scribbler robot with Fluke

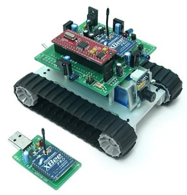
Fig. 2 SRV-1 robot (ARM-7 proc.)

The readily existing drivers are written in C/C++ and the existing simulators use Open GL. For instance, the Myro library [7] already provides a driver implementation for the Scribbler written in Python, and Surveyor Corp. provides its own implementation of drivers, firmware, and console applications to control the Surveyor robots. In addition, Prof. Sklar and her research colleagues have already adapted their own C/C++ libraries to interface the Scribbler and the SRV-1 as explained in [8], and this software can be obtained freely at [9].

II. Player/Stage

[1] "The Player Project creates Free Software that enables research in robot and sensor systems. The Player robot server is probably the most widely used robot control interface in the world. It supports a wide variety of hardware, also C, C++, and Python programming languages are officially supported." Stage is Player's 2-D simulation backend, and Gazebo is its 3-D simulation environment. [10] explains thoroughly the real reusability and abstraction layer that the Player/Stage tools can provide for the robot-application programer. Besides, the ability to simulate algorithms for autonomous robot navigation, and other sensing behaviors such as distributed perception – in the case of multiple-robot environments – are just a few of the numerous reasons why we decided to focus on Player/Stage.

At Brooklyn College, there are already courses such as Introduction to Artificial Intelligence (CIS 32) [11], that use Player/Stage to simulate AI fundamentals. Now, the task at hand is to implement drivers for these educational robots so they can be used with the Player/Stage API and thus take advantage of the abstraction from hardware complexity and the reusability of code in order to develop reusable controllers (brains) for these and other robots.

III. Surveyor Plugin Driver for Player

[12] and [13] introduce the key concepts involved in the process of writing Player/Stage drivers for other robots besides the ones that currently exist in the official releases of the Player package, like in the case of the Roomba robot, for example.

A. About the Surveyor SRV-1

Fig. 2 shows a SRV-1 robot, which resembles a small military tank with treads. Only the first generation of the SRV-1 robots are available in the lab. These have an ARM-7 processor on-board, but due to this limited processing power, all of the data collected by its sensing devices are generally processed by the Player server running wirelessly on an off-board computer. Thus, the SRV-1 communicates serially with the controlling computer via an XBee radio dongle. The SRV-1 also has a low-resolution camera and four infra-red sensors. This 4-wheeled robot has two treads on each side that improve its traction on rescue arenas. We want to use the camera as its main sensing device to interface with Player/Stage for SLAM (simultaneous localization and mapping) capabilities.

B. Inheriting from Player's Driver Class

The driver acts as a proxy between the robot's hardware interfaces and the function calls to the Player server. In order for Player to be able to understand such driver implementation, it must inherit from Player's "Driver" class. For portability reasons, the SRV-1 driver is being implemented as a plugin driver rather than as a statically linked one. A plugin driver is compiled as a shared-object that is dynamically linked to the Player server's driver table at run-time. We specify the use of this plugin driver through the configuration file that is passed when we start up Player from the command line, as following:

```
$   player surveyor.cfg
```

The *surveyor.cfg* in Fig. 3 is an example configuration file with various options, in which the shared-library for this plugin driver is named `libSurveyor_Driver.so`

```
driver
(
  name "surveyor"
  plugin "libSurveyor_Driver.so"
  provides ["position2d:0" "camera:0"]
  port "/dev/ttyUSB0"
  image_size "320x240"
)
```

Fig 3. Example of *surveryor.cfg* file for a plugin driver

Upon loading a plugin driver, the Player server calls the "player_driver_init" method, whose main purpose is to initiate the registration of the driver into the given driver table by calling the "Surveyor_Register" function. By passing the corresponding configuration file to this registering function, the name of the driver – specified to be "surveyor" in this case – is then passed to the driver class factory function "Surveyor_Init", which returns a pointer (as a generic Driver*) to the new instance of this driver.

After the driver has been instantiated and its central thread has been started by the Setup() function, the remaining processing is handled by the Main() method, which primarily consists of a loop that usually consists of 3 steps:

1. sensor and state data are collected, then
2. collected data are published to the relevant devices,
3. and finally, incoming messages are processed.

A brief list of the Driver class' methods that need to be implemented is given in Table 1.

TABLE I
INHERITED "DRIVER" CLASS FUCTIONS

Name	**Brief Description**
Public Member Functions:	
Surveyor (ConfigFile *cf, int section)	Constructor for the Surveyor multi-interface driver.
int Setup ()	Set up the device and start the device thread by calling StartThread(), which spawns a new thread and executes Surveyor::Main(), which contains the main loop for the driver.
int Shutdown ()	Shut down the device.
int ProcessMessage (QueuePointer &resp_queue, player_msghdr *hdr, void *data)	Message handler that sends a response if necessary using Publish(). This function is called once for each message in the incoming queue.
Private Member Functions:	
virtual void Main()	Main "entry point" function for the driver thread created using StartThread() within the Setup() function
Server and Plugin-Specific Hooks	
int player_driver_init (DriverTable* table)	Initiates the registration of the driver into the given driver table.
void Surveyor_Register (DriverTable *table)	Driver registration function that adds the driver into the given driver table
*Driver** Surveyor_Init (ConfigFile *cf, int section)	Factory creation function that instantiates the Driver

The communication procedures for the SRV-1 are defined outside the inherited driver class. Although this separation is not necessary, it purely allows for a more object-oriented programming practice. Most of the control protocol for the SRV-1 are single character commands that are sent to the robot and acknowledged back with a '#' character followed by the original command. For example, the command: 'Mabc' implies direct motor control, so the acknowledge response would be just '#M'

In this example, the 'abc' parameters are sent as 8-bit (1-byte) binaries that specify the following:

a=left speed, b=right speed, c=duration*(10milliseconds)
where speeds are 2's complement 8-bit binary values, so
0x00 through 0x7F are used to move forward,
0xFF through 0x81 are used to move in reverse,

Thus, the decimal equivalent of the 4-byte sequence (in hex notation) 0x4D 0x32 0xCE 0x14 is equivalent to 'M' 50 -50 20, which means "rotate right at 50% speed for 200ms". A duration of 00 would imply to be infinite. Hence, the 4-byte sequence 0x4D 0x32 0x32 0x00 = M 50 50 00 drives the SRV-1 forward at 50% speed indefinitely.

C. *Results*

The Player driver for the surveyor SRV-1 was initially tested with Player's CLI (command line interface) utilities such as *playerjoy* and *playercam*. Obviously, any other controller for Player can do the job as long as the interfaces are properly provided in the configuration file as indicated previously. Thus, the robot is capable to move around in various directions on the plane, but for now it does not support odometry to tell its exact 2-D position in the physical world because these particular SRV-1 robots' actuators are only regular DC motors instead of the more precise servo-motors that can provide exact positioning data. We also know that the IR sensors are not reliable enough to use them as range-finding interfaces. For now, this implemented Player/Stage driver lacks of the infrared interface. Also, the camera's frame rate appears very low at the moment (around 1 fps), yet it is known to be capable of faster rates.

IV. SRVjoy: a Player GUI Console to Control the SRV-1

Since we wanted a GUI (graphical user interface) that would allow a human operator to drive the robot and take snapshot images from the camera as well as to be presented with a live-video feed that can, eventually, be integrated with SLAM capabilities – to be added to the "Stage" simulator in future projects – *SRVjoy* is a GUI console written with Nokia's QT, an open-source and cross-platform GUI library.

SRVjoy can also control any other robot that has Player drivers and provides similar interfaces. For now, it only supports position2d and camera interfaces, and can connect to a robot through Player's default port number "6665" being served on the "localhost". Thus, the operator can drive the robot around (controlling both linear and angular speeds), and take snapshot images from the robot's camera. A video feed – native to the console – has not been implemented, yet *playercam* can be used externally at the same time SRVjoy is running.

The navigation can be done through the console buttons in the GUI shown in Fig. 4, or by using some assigned keys in the keyboard (The numeric keypad is the preferred choice at the moment).

The SRVjoy console operates in the following way:

- All control buttons have icons that show their moving directions. By pressing a direction button, the robot moves in that direction for as long as the button remains pressed.
- Speed (linear and angular) can be controlled with the sliders.
- The camera button will take a snapshot of the current camera view and save it as camera####.jpg in the root folder of the executing program.

  NOTE: There is some delay for snapshots (the obot has to focus first, and then shoot), and sometimes the order of pictures appears shifted.
- Numkeys also resemble the layout of the console buttons, so they can be use to control the robot in a similar manner.

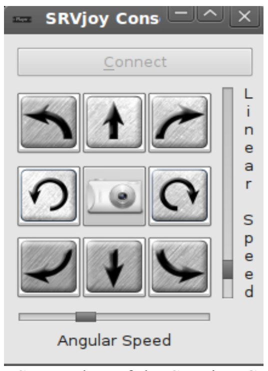

Fig 4. Screenshot of the SRVjoy Console

V. Conclusions

At first, the research experience took me in several directions until getting acquainted with the variety of software that can be employed to interact with these small robots. As explained throughout this report, Player/Stage was the preferred tool to dive into, as it provides the right abstraction from the hardware details and presents a uniform API to the user. As a Computer Engineering major, I found myself deeply interested in swimming across this abstraction layer - where hardware meets software – by writing drivers that can fill this void at the valley of the unsupported robots and Player, "One Hell of a Robot Server"

There is more to be done to achieve a more polished functionality of the SRV-1 driver and the SRVjoy console, and I feel that leaving the source open at hands of the GNU General Public License can assure that this goal is reached at some point by the joined efforts of the open-source community. All source code and documentation (at the time of this writing) is freely available on my DREU Project's website at http://agents.sci.brooklyn.cuny.edu/~cjaramillo/project.php

Acknowledgment

Thanks to John Cummins for his devoted guidance and the work he has done –working in conjunction with professors Sklar and Parsons– for the C/C++ libraries for the Scribbler and SRV-1 robots. The clarity of his comments in the source code allowed me to understand the underlying communication procedures between the robot's firmware and the higher-level libraries. Also, thanks to Michael Jansen for the original implementation of the surveyor driver for Player/Stage that can be informally found in the Surveyor forums.

References

[1] (2009) The Player Project website. [Online]. Available: http://playerstage.sourceforge.net
[2] Exploring Robotics (CC 30.03) course website [Online]. Available: http://agents.sci.brooklyn.cuny.edu/cc30.03/
[3] Bridges High School Workshop website. [Online]. Available: http://bridges.brooklyn.cuny.edu/index.php/home
[4] Lego Education website [Online]. Available: http://www.legoeducation.com
[5] Surveyor Corporation website. [Online]. Available: http://www.surveyor.com/
[6] Institute for Personal Robots in Education (IPRE) website. [Online]. Available: http://roboteducation.org

[7] Scribbler's Myro [Online]. Available: http://wiki.roboteducation.org/Myro_Reference_Manual

[8] John Cummins, M. Q. Azhar, and Elizabeth Sklar, "Using Surveyor SRV-1 Robots to Motivate CS1 Students," *AAAI.org*, 2008

[9] Surveyor Robot Software from the Agents Lab at Brooklyn College. [Online]. Available: http://agents.sci.brooklyn.cuny.edu/robotics.edu/bcsoftware.php

[10] Brian P. Gerkey and Richard T. Vaughan, "Really Reusable Robot Code and the Player/Stage Project." *Software Engineering for Experimental Robotics Springer Tracts on Advanced Robotics*, Springer, 2006.

[11] Introduction to Artificial Intelligence (CIS 32) course website. [Online]. Available: http://www.sci.brooklyn.cuny.edu/~parsons/courses/32-spring-2009/

[12] Player/Stage Drivers: Writing a Player Plugin from the Penn State Robotics Encyclopedia RoboWiki. [Online]. Available: http://psurobotics.org/wiki/index.php?title=Player/Stage_Drivers

[13] Jonas Fonseca and Bue Petersen, "Writing Player/Stage Drivers: a howto for ERSP Player Driver Source Package," From the project: Player/Stage - *Player driver implementation for ERSP*, 2006