The Magellan is Back
Player/Stage on the Magellan Pro Robot
SNeRG Technical Note 40

Timothy J. Burns
Department of Computer Science and Engineering
University at Buffalo, the State University of New York
Buffalo, N.Y. 14260-2000
tjburns@cse.buffalo.edu

December 14, 2007

Abstract

This report details our experience of running Player/Stage, an open source application available on the net for developing robot control programs, on the Magellan Pro robot. The Player/Stage software is a networked robot/sensor device interface which when configured using various driver files can be used to communicate with virtually any type of mobile robot system. For our purposes, however, we will only be concerned with the Magellan, and this report outlines the most important knowledge that we acquired for future research projects involving the Magellan and the Player software.

1 Introduction

Our main goal for this project was to implement a SNePS agent on IRobot's Magellan Pro robot. The Magellan, which has received various treatments over the years by numerous SNeRG [10] researchers [5, 6], had been completely revamped as far as it's internal OS and architecture were concerned so it became a task of basically starting from virtually scratch. We decided to use the open source Player/Stage [1] software to provide the mid-level link between a SNePS agent and the robot itself. As time went on, however, due to numerous potholes and roadblocks, it because clear that the main goal was going to have to change to simply getting the platform stable enough for further work. Much of the reason for this is due to the lack of documentation for both the Magellan itself, and the Player/Stage software which, despite being very useful, was a considerable challenge to sort out. My own personal lack of knowledge in this particular area also provided a (welcome, but frustrating) challenge of its own. What follows should be enough for future researchers to take-in and use in implementing higher-level agents on the Magellan.

2 The Magellan Robot

The Magellan Pro [2,3] was developed by the IRobot Corporation [4] and has been with the Department of Computer Science and Engineering since 2003. Unfortunately, since developing the Magellan the IRobot Corporation has entered other markets and stopped producing and supporting the Magellan so to say support is limited would be an understatement. However, the robot does provide a lot of different and interesting functionality so it still remains a worthwhile research tool in cognitive science and robotics.

This particular Magellan is equipped with 16 sonar sensors, 16 IR (Infrared) sensors, and 16 tactile ("bumpers") sensors. Internally these 16 sensor sets are split into two separate units each of which is responsible for 8 sensors (8 for the front and 8 for the back). Our robot also comes equipped with a SONY pan/tilt/zoom camera and wireless capability so the robot can operate fully-autonomously as long as there is a wireless connection within range (unless its client software is running internally). The robot also has the capability of operating under joystick control. The Magellan is currently running a recent release of RedHat Linux as an operating system.

2.1 Starting the Magellan

Getting the Magellan started turns out to be a strange endeavor. For reasons unknown to me there is no simply on/off toggle switch. Instead, one must physically open up the rear panel on the robot (there are two silver release switches on the right side looking from the back to do this) and unplug the three wires at the roof of the frame (two grey wires with a red Ethernet cable in the middle). After unplugging one must then plug them all back in rapidly and the system will boot. Again, this is very strange, but I don't know of a better way.

Another note on booting is that if you wish to use joystick control at any point during the lifespan of this current boot you must have the joystick plugged into the serial port (on to of the robot) before booting. Otherwise the robot will not recognize the presence of a joystick.

2.2 Charging the Magellan

The Magellan comes with a massive charging unit which plugs into a wall outlet and is then twisted into place at the charge port on top of the robot. There a few ***very*** important things to note about charging the Magellan. Whenever you are charging the robot you must not leave the robot in a powered off state otherwise the battery will overcharge and ooze out which would most likely destroy the robot. Instead you should either unplug the charging cable or leave the robot powered on. In this case the internal controls will take control of regulating the voltage into the battery when it becomes fully charged in order to not overcharge it.

2.3 The main console

The main console is displayed on a small LCD on the top of the robot and contains a number of sub menus. When highlighting the selection of choice one can push this knob in to enter that submenu. Most of these menus contain simple test environments for raw sensor data. In these environments one can turn on and off various sensors and get raw readings from the different banks onboard the robot. Of particular interest to the reader will be the Joystick Drive and Host Console menus. The former allows you to enter a joystick input console where the robot can be controlled manually. This manual control is particularly useful if a client program goes amiss and runs the robot into a wall! The host console will display booting information for RedHat when the OS is loading. After booting, this console will not display anything too useful since login to the robot will most likely be done remotely. This menu does contain the option to power off the robot, however, which should only be done after the operating system is booted down.

2.4 Remote login and other information

In order to access the internal OS of the robot remotely the robot itself must be booted up (as described above) and you must have access to a UNIX terminal on any machine with network access. At the prompt type,

pollux {~} > ssh tjburns@irobot.cse.buffalo.edu

The username would be replaced with your own in the above command. In order to obtain an account on the robot itself you should contact Christian Miller [cwmiller@cse.buffalo.edu] from CSE-Consult. Although most packages (including Player) have been installed into the system path, most can be found in the *~/packages* directory.

3 The Player/Stage Software

The Player project is a collaborative open source project which is described as providing an abstraction layer for robotic devices. Player defines a set of standard interfaces which specify the different ways of interacting with certain classes of devices. By providing this kind of abstraction Player control programs can theoretically, and within reason, have the same effect on robots that are different morphologically. In addition to this portability, Player is also transportable. That is, a player server can be run using sockets and a specified driver for the hardware involved, and any client programs may then be run from a different location over the network through the specified sockets. This allows for increased autonomy (particularly for us since the Magellan has a wireless adapter) and easier development of control programs. Development of control programs is made simple by the provision of libraries in common languages: C, C++, Java [7], Python [9], and Common LISP [8]. Although it is not discussed here, the Common LISP libraries might be of particular interest in further development on SNePS agents on the Magellan. Further information on Player/Stage and its many contributors and accessory libraries may be found at the project website. We describe here the basics necessary to develop new control program libraries for the Magellan using the built in C++ library.

3.1 The config file

In order to start an instance of the Player server on the robot itself one must simply type the following at the RedHat shell

[tjburns@irobot ~]$ player <config_file>.cfg –p <port_number> &

The port number is optional and defaults to 6555. The configuration file is necessary to provide Player with the information it needs to communicate with the physical hardware of the Magellan. The exact formation of these configuration files looks something like

```
driver
(
  name "p2os"
  provides ["odometry:::position2d:0"]
  port "/dev/ttyS0"
)
driver
(
  name "sicklms200"
  provides ["laser:0"]
  port "/dev/ttyS1"
)
```

for a robot with simple drive motors and lasers onboard. There are many default configuration files installed automatically that can be found in

/usr/local/share/player/config on the Magellan. The Magellan uses the rflex driver to control all of its onboard motor controls and sensors. After a lot of experimenting and messing with these driver files we have achieved what appears to be a suitable config file for our particular robot. The contents of the file can be seen in Appendix A. Currently this file is stored in the *~/packages* directory.

3.2 Client Programs

As discussed, layer is supported by numerous client libraries some of which are built in to the existing installation. Here we discuss the built in C++ client libraries. Numerous example files for this library can be found in the *~/packages/player-2.0.2/examples/libplayerc++* directory. The general structure of a client program is given by the following outline:

1 Connect to robot by constructing a PlayerClient object
2 Create devices that are to be used in the program by requesting them from the PlayerClient object.
3 while(someConditionToFinishIsNotTrue)
4 {Read the data from devices based on received data determine actions}

The Magellan provides three different sensor types (IR, Sonar, Bumper) along with drive control and camera input. These devices are accessed in a control program through various Proxies which are requested in step 2. These proxies include:

- *BumperProxy*
- *SonarProxy*
- *Position2dProxy*
- *IRProxy*

An example C++ client program is listed in Appendix B. This program is a very simple wall following procedure that attempts to maintain a certain buffer distance to the left wall at all times in an infinite loop.

3.3 Compiling

Although there are ways provided through Player to have complicated projects compiled through makefiles, these client programs can also be compiled very simply using the following command:

```
g++ -o exeName `pkg-config --cflags playerc++` clientProgramr.cc `pkg-config --libs playerc++`
```

In order to run this executable, player must be running on a specified port. It is a good idea to incorporate command line arguments into your programs so as to specify ports other than the default (6555) to your program.

4 Future Work

During the course of this project we have been able to produce a good foundation from which to build upon by sorting out some of the most challenging low-level details in the robot itself and the sometimes poorly documented Player software. Although the actual functionality of the client program developed for the Magellan is somewhat crude, it does provide a template from which development of a more general control abstraction in C++ (or LISP if the ACL client libraries are used!) for the Magellan. This kind of program should include functions for general tasks like the following:

- FollowWall
- FindNextDoor
- SenseSurroundings

These tasks would provide the input from the world from which a higher reasoning system like SNePS would be able to operate and provide control commands back to the robot. In this way SNePS could function like a brain for the physical body of the Magellan.

Another note of interest is that the Java client is currently installed on the system and is located in ~/JavaClient. This directory structure contains many examples that could provide quite useful for developing Java clients.

Overall, this project has been a great learning experience in areas that I have not been familiar with, and I feel that this, along with this manual of sorts, will provide a much easier point to start from for future work.

Appendix A

Note: Comments start with #

```
driver
(
name "rflex"
provides ["sonar:::sonar:0" "position2d:0" "ir:0" "power:0" "bumper:0"
]
rflex_serial_port "/dev/ttyS1"
m_length 0.5
m_width 0.5
odo_distance_conversion 31387
odo_angle_conversion 4654.2
default_trans_acceleration 0.2
default_rot_acceleration 0.2

# bumpers
rflex_bumper_style "bit"

# Sonar options
range_distance_conversion 1.474
sonar_age 1
sonar_echo_delay 30000
sonar_ping_delay 1250
max_num_sonars 64
num_sonars 16
num_sonar_banks 2
num_sonars_possible_per_bank 16
num_sonars_in_bank [8 8]
mrad_sonar_poses [4.320 76.537 184.776 3.927 141.421 141.421 3.534
184.776 76.537 3.142 200.000 -0.000 2.749 184.776 -76.537 2.356 141.421
-141.421 1.963 76.537 -184.776 1.571 -0.000 -200.000 1.178 -76.537 -
184.776 0.785 -141.421 -141.421 0.393 -184.776 -76.537 0.000 -200.000
0.000 -0.393 -184.776 76.537 -0.785 -141.421 141.421 -1.178 -76.537
184.776 -1.571 -0.000 200.000 ]

# IR
pose_count 16
ir_min_range 0.100
ir_max_range 0.800
rflex_bank_count 2
rflex_banks [ 8 8 ]
rflex_ir_calib [
4000 -2
4000 -2
4000 -2
4000 -2
4000 -2
4000 -2
4000 -2
4000 -2
4000 -2
4000 -2
4000 -2
4000 -2
```

```
4000 -2
4000 -2
4000 -2
4000 -2
]
# y, x, theta
poses [
0.370 -0.153 5.890
0.283 -0.283 5.498
0.153 -0.370 5.105
-0.000 -0.400 4.712
-0.153 -0.370 4.320
-0.283 -0.283 3.927
-0.370 -0.153 3.534
-0.400 0.000 3.142
-0.370 0.153 2.749
-0.283 0.283 2.356
-0.153 0.370 1.963
0.000 0.400 1.571
0.153 0.370 1.178
0.283 0.283 0.785
0.370 0.153 0.393
0.400 -0.000 -0.000
]
)

driver ( name "sonyevid30"
      provides [ "ptz:0" ]
      port "/dev/ttyS2" )
```

Appendix B

```
#include <libplayerc++/playerc++.h>
#include <iostream>
#include "args.h"

using namespace PlayerCc;


int main(int argc, char **argv)
{
  // Program Constants
  double min_front_dist = 300.0;
  double really_min_front_dist = 100.0;
  int following = 0;

  parse_args(argc,argv);

  try
  {

    // Subscribe to the required Sensor Proxies
    PlayerClient robot (gHostname, gPort);
    Position2dProxy pp (&robot, gIndex);
    SonarProxy sp (&robot, gIndex);

    std::cout << robot << std::endl;

    // Take the break off
    pp.SetMotorEnable (true);

    double newspeed = 0.0f, newturnrate = 0.0f;
    // Find the wall:
    while(following != 1)
    {

      // Read from all the sensors
      robot.Read();

      // Initialize the default speed
      newspeed = 0.200;

      // sp[] is the sonar sensor array.
      //   indexed from 0-15.
      //   [0-7] in rear
      //   [8-15] in front
      if(((sp[10] < really_min_front_dist) || (sp[11] < really_min_front_dist)))
        {
          // Backup - too close!
          newspeed = -0.200;
        }
      else if(((sp[10] < min_front_dist) || (sp[11] < min_front_dist)))
        {
          // Wall ahead, turn right
          newspeed = 0;
          newturnrate = -0.5;
        }
      else if ((sp[8] < min_front_dist) || (sp[9] < min_front_dist))
        {
          // Leftside now lined up to the wall so stop and continue below
          newspeed = 0.2;
          newturnrate = 0;
          following = 1;
        }

      // write commands to robot's drive control
      pp.SetSpeed(newspeed, newturnrate);
    }

  // Now follow the wall:
```

```
  for ( ;; ) {
    // Read from the sensors
    robot.Read();

      newspeed = 0.200;

      if(((sp[10] < really_min_front_dist) || (sp[11] < really_min_front_dist)))
        {
          // Backup - too close!
          newspeed = -0.200;
        }
      else if(((sp[10] < min_front_dist) || (sp[11] < min_front_dist)))
        {
          // Wall ahead, turn right
          newspeed = 0;
          newturnrate = -0.5;
        } // ---------------- Keep the left wall within a certain threshold: -------------
      else if ((sp[8] < min_front_dist) || (sp[9] < min_front_dist))
        {
          newturnrate = -0.5;
        }
      else if ((sp[8] > min_front_dist) || (sp[9] > min_front_dist))
        {

          newturnrate = 0.5;
        }

      // write commands to robot's drive control
      pp.SetSpeed(newspeed, newturnrate);
  }
  }
  catch (PlayerCc::PlayerError e)
  {
    // There has been an error, so report:
    std::cerr << e << std::endl;
    return -1;
  }
}
```

References

[1] Player/Stage Source Forge Homepage
http://playerstage.sourceforge.net/

[2] Magellan Pro Compact Mobile Robot User's Guide

[3] IRobot's Mobility Manual
http://www.cse.buffalo.edu/~shapiro/Courses/CSE716/MobilityManRev4.pdf

[4] IRobot.
http://www.irobot.com

[5] Trupti Devdas Nayak, Michael Kandefer, and Lunarso Sutanto, Reinventing the Reinvented Shakey in SNePS, SNeRG Technical Note 36, Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY, April 6, 2004.

[6] Isadore Dinga Madou, GLAIR Agents on the iRobot Magellan Pro Robot, SNeRG Technical Note 37, Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY, December 17, 2004.

[7] JavaClient
http://java-player.sourceforge.net/

[8] LISP Client
http://www-robotics.cs.umass.edu/~bburns/software/player-lisp.html

[9] Blank, D.S., Kumar, D., Meeden, L., and Yanco, H. (2005) The Pyro toolkit for AI and robotics. To appear in AI Magzine.

[10] SNePS Research Group Homepage
http://www.cse.buffalo.edu/sneps/