System for robotic e-learning

SyRoTek

Player Proxies and configuration used in PAR course

ver. 1.0

Intelligent and Mobile Robotics Group
The Czech Institute of Informatics, Robotics and Cybernetics
Czech Technical University in Prague

Contents

Chapter 1

Introduction

In this documentation the basic Player C++ proxies are described as well as the basics of the Player's configuration files. Beside these topics the SND driver used to control the robot is briefly described. It should help to the new user to get familiar with these concepts as fast as possible.

The official documentation for more detailed information is of course available. List of all the proxies offered by Player is stated in [1] and simple overview of how to create the configuration file is in [2]. But please be aware of the Player's version you are using and pay attention to the programming language the documentation is related to (especially C vs. C++).

Chapter 2

Proxies

In the Player's convention a proxy is a class, which wraps the functionality of a specific device and provides high level methods to control the device or read required information. Thus the user doesn't have to take care of any low level communication protocols, device connection (this is specified in the configuration file) and other necessities related to the implementation of the device into the robotic system, e.g. a sensor such a laser scanner.

During the Practical Robotics course only two main proxies are used - LaserProxy, Position2dProxy. The classes of these proxies are derived from one ancestor class - ClientProxy. All these three classes are described below.

2.1 ClientProxy

This class precedes all later described classes. It contains basic proxy information such a device ID and name, provides interface to set and get property of the device and a data status. The data status is probably the most important functionality of the class for us.

Method `bool IsValid()` can be used to check if the data were successfully received. It returns true in case that any data from the device were received, false is returned otherwise. Return value is fully handled by the ClientProxy.

Contrary a method `bool IsFresh()` is user-controlled. Whenever new data are received, this method automatically returns true. But when the data are read by the user code, a flag, that the data are already processed, can be set by method `void NotFresh()`. After calling `NotFresh()`, the method `IsFresh()` returns false until new data are received. This tells us, for example, in next iteration of our algorithm, that we have already read the data and no new reading is available since then.

2.2 LaserProxy

LaserProxy represents the laser scanner attached to the robot. Every laser scanner has some angle of view, angular and distance resolution and number of points in one scan. All these information can be retrieved through the LaserProxy.

Number of the points in a scan is returned when calling method `uint32_t GetCount()`. The angle of view can be determined from values returned by `double GetMinAngle()` and `double GetMaxAngle()`, which return scan range of the latest data in radians. Scanner's angular resolution is available via `double GetScanRes()`, result is in radians, similarly the distance resolution in millimeters is returned by `double GetRangeRes()`.

When a new measurement is received, the distance of one point can be read using `double GetRange(uint32_t aIndex)`, where the parameter aIndex is index of the point. Indices begin with 0 at the left and ends with `GetCount()` - 1 at the right. Except `double GetRange(uint32_t aIndex)` also the operator `[]` can be used to read the distance of the point (e.g. myLaser[index]). Bearing of the point in radians, i.e. the relative angle from the sensor's heading with positive sense of rotation, is returned by `double GetBearing(uint32_t aIndex)`. You can also check the intensity of returned signal in given point by `int GetIntensity(uint32_t aIndex)`.

Beside these basic methods there are several more methods remarkable, which can simplify your work during the course. First of the methods is `player_point_2d_t GetPoint(uint32_t aIndex)`. Point returned represents the measured point coordinates with respect to the sensor position (i.e. coordinates are computed from bearing and distance of the point). The method returns instance of a simple structure `player_point_2d_t` containing two public members `px` and `py` representing the coordinates of a point in Cartesian 2D space.

Another useful method is `player_pose3d_t GetPose()` which gives the position of the laser scanner with respect to its parent object - in our case the robot. Structure `player_pose3d_t` is similar to the mentioned `player_point_2d_t`, but moreover contains the third coordinate and rotation of the object - components `pz`, `proll`, `ppitch`, `pyaw`.

In the end we will describe two methods interesting mainly during playing with robot at the beginning of the course. These are `double GetMinLeft()` and `double GetMinRight()` which allow to get minimal distance of the measurement on the left and right side of the scan.

2.3 Position2dProxy

Position2d proxy provides information about position and speed as well as is allows to control these properties. In the PAR course code this type of proxy is not used only to refer a position of the robot, but also to control the robot with the SND driver, which provides one Position2dProxy instance. The SND driver's detailed description is below.

Position can be retrieved by methods `double GetXPos()`, `double GetYPos()` and `double GetYaw()`. Units used in course are meters and radians for an angle. Therefore to use them in a map, a conversion is necessary. Similarly to the position, the speed can be read by methods `double GetXSpeed()`, `double GetYSpeed()` and `double GetYawSpeed()`, results are again in meters or radians per second.

It is also possible to change the odometry value with `void SetOdometry(double aX, double aY, double aYaw)` or reset it with `void ResetOdometry()`, which is equal to `SetOdometry(0, 0, 0)`.

User can control the robot's speed directly using the methods `void SetSpeed(double aXSpeed, double aYSpeed, double aYawSpeed)`, `void SetSpeed(double aXSpeed, double aYawSpeed)` or `void SetSpeed(player_pose2d_t vel)`. All the mentioned methods are the same except their input arguments. For the Syrotek system y-speed has no meaning since the robots in the arena are non-holonomic, therefore the option with only X and yaw speed is recommended when a direct speed control is used. To move the robot, motors have to be enabled first with `void SetMotorEnable(bool enable)`.

To change the robot's position use the method `void GoTo(player_pose_2d pos)` or `void GoTo(double aX, double aY, double aYaw)` of the SND driver. After calling this method the driver takes control of the robot and drives it to the required position.

2.3.1 SND driver

SND stands for Smooth Nearness Diagram navigation. It extends the functionality of ND (Nearness Diagram) driver. The algorithm is designed for the non-holonomic robots moving in tight spaces, i.e. fits the needs of the Syrotek system. Detailed description of the algorithm itself was published in paper [4].

SND driver can be found in both simple and real configuration files. It provides one position2d proxy - this is available in user application. But it requires two position2d proxies - one as a source of the robot's position and one as a sink for speed commands.

It has several configuration options where the parameters of the robot can be set, its maximal speed and position error. Parameters are described in the following list (description is taken from [5]):

- robot_radius - The radius of the minimum circle which contains the robot [m]
- min_gap_width - Minimum passage width the driver will try to exploit [m]
- obstacle_avoid_dist - Maximum distance allowed from an obstacle [m]
- max_speed - Maximum speed allowed [m/s]
- max_turn_rate - Maximum angular speed allowed [rad/s]

- goal_position_tol - Maximum distance allowed from the final goal for the algorithm to stop. [m]
- goal_angle_tol - Maximum angular error from the final goal position for the algorithm to stop. [m]

During the debugging of your application you should pay attention especially to the parameters concerning the maximum speed.

Chapter 3

Configuration

When starting Player the configuration file has to be specified. It tells the Player which drivers to load, how to set them up and how to create interface for client application. Simple configuration files are used during the course - files simple.cfg and real.cfg included in the basic source code provided for the semestral tasks.

It consists of several driver definitions. Each driver means a new instance of the specified driver started after the Player starts. Drivers are described basically by two mandatory options - name and provided attachment points. The `name` specifies the name of the driver to be loaded (it is not user defined label but name well known to the Player). Option `provides` tells the Player what proxies are offered by this driver to the other drivers or client application. It consists always from the proxy name and unique index (index have to be unique for given proxy type).

Moreover it is possible to define the driver by additional option keywords. If the driver expects an input from other driver, it can be specified using `requires` where expected proxy is stated. To allow using user hardware the `plugin` option can be defined with a name of a shared library containing the driver. As already mentioned at SND driver, each driver can have itself its own options (e.g. maximum speed, size of the robot, etc.) specific to each type of the driver.

Bibliography

[1] Player 3.0.2 documentation - list of C++ proxies. [Online]. http://playerstage.sourceforge.net/doc/Player-3.0.2/player/group__player__clientlib__cplusplus__proxies.html

[2] Player 3.0.2 documentation - Writing configuration files. [Online]. http://playerstage.sourceforge.net/doc/Player-svn/player/group__tutorial__config.html

[3] Player 3.0.2 documentation - ClientProxy. [Online]. http://playerstage.sourceforge.net/doc/Player-3.0.2/player/classPlayerCc_1_1ClientProxy.html

[4] Durham, J. ; Bullo, F. "Smooth Nearness-Diagram Navigation" 2008, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008, 690-695

[5] Player 3.0.2 documentation - SND driver. [Online]. http://playerstage.sourceforge.net/doc/Player-svn/player/group__driver__snd.html