User manual

Links

The source code is available at: http://mytestbed.net/projects/robot/repository

Compiling the demo

The demo this document is alluding to is a command line demo using the XML interface of the developed navigation API. The demo source code is in `PROJECT_DIRECTORY/navigation/demo_v4.cc`

Before compiling the demo, you must first compile the two libraries it relies on. **Libirobot-create** and **ARToolKit**.

Compiling libirobot-create

```
$ cd PROJECT_DIRECTORY/libirobot-create/
```

If first compilation

```
$ autoreconf --install
$ ./configure
$ make
```

else

```
$ make
```

Compiling artoolkit

```
$ cd PROJECT_DIRECTORY/ARToolKit
```

If first compilation

```
$ ./Configure (answer: 3, y (x86), n, n)
```

```
$ make
```

else

```
$ make
```

Compiling the demo

```
$ cd PROJECT_DIRECTORY/navigation/
$ make demo_v4
```

Running the demo

Directly on the laptop connected to the robot

First you need to connect the laptop to the robot and hit the robot's power button. Then on the laptop do the following:

```
$ cd PROJECT_DIRECTORY/navigation/
$ python pySerialOpen.py (needed once to open the serial port)
$ gdb ./demo_v4
$ run
```

Using a SSH connection

You can also run the software remotely from another PC. As the software displays the video when running (to disable this option see next section), you will need to specify the display to be kept on the remote laptop.

```
$ ssh login@laptop_ip (the ip of the laptop connected to the robot)
$ export DISPLAY=:0.0 (set display on the remote laptop)
$ cd PROJECT_DIRECTORY/navigation
$ gdb ./demo_v4
$ run
```

Disabling video display

You may want to run the navigation software whithout seeing the video stream of the camera. To do so you will need to change a variable in the source code and recompile the demo.

Open the file `PROJECT_DIRECTORY/navigation/navigation-vision-system-bridge.c` and change at line 24 `#define DISPLAY 1` to `#define DISPLAY 0`

The recompile the demo:
```
$ make demo_v4
```

XML interface of the navigation API

The navigation API accepts commands in XML Format. The class in charge of handling the XML format is implemented in `navigation-api-xml-interface.cc` file.

The test file `navigation-api-xml-interface-test.cc` provides an example of how to use the XML interface.

The XML interface exposes two methods:

- `string readXml(string xml)`
- `string readXmlFile(string filename)`

`readXml` takes as a parameter a string containing the command in XML whereas `readXmlFile` take as a parameter the path to an XML file containing the command.

Both of those two methods return a string in XML format when the supplied command has to return a result otherwise the returned string is empty.

The set of available commands and their XML format is given in the next section.

Navigation commands

Each command is represented by a **XML tags**. All the **coordinates** and **dimensions** are expressed in **mm**, the **angles** are expressed in **degrees** and the **XML interface** is **case sensitive**.

Below is an example to get the current state of the robot.

```
NavigatorXmlInterface robot;
string state = robot.readXml("<getState />");
std::cout << state << std::endl;// displays <state value='IDLE' />
```

The navigation commands are described in XML format. Simple commands are represented by self-closing tags such as `<getState />` while others are represented by nested tags such as `<setJournet></setJourney>` and `<setMaps></setMaps>`.

Below is the list of the commands in XML format.

```
<getState />
<getMaps />
<clearMaps />
<resetMovement />
<resume />
<pause />
<isAtLocation x='0.0' y='0.0' /><!-- in mm -->
<isAtLocation x='0.0' y='0.0' thresh='30' /><!-- in mm -->
<isAtMarker name='name' />
<getLocation />
<setMaps>
      <map name='WS-L4-19' x='0' y='0' z='0'><!-- in mm -->
            <!-- markers -->
            <marker name='Hiro' x='0' y='0' z='0' size='96' angle='90' />
            <marker name='Kanji' x='152' y='-31' z='0' size='200' angle='0' />
            <!-- links between markers -->
            <path between='Hiro' and='Kanji' />
      </map>
      <map name='WS-L4-XX' x='100' y='2000' z='0'>
            <marker name='4x4_3' x='0' y='0' z='0' size='194' angle='90' />
            <marker name='4x4_4' x='-1981' y='1531' z='0' size='194'
angle='90' />

            <path between='4x4_3' and='4x4_4' />
      </map>
</setmaps>
<gotoMarker name='Hiro' />
<getNearestMarker />
<gotoNearestMarker />

<setJourney>
      <gotoMarker name='Hiro' />
      <gotoMarker name='Kanji' />
      <gotoNearestMarker />
```

```
</setJourney>
<locate timeout='10'/><!-- in seconds (optional) -->

<!-- additional features -->
<getName />
<setName name='name' />
<setMarkerDetectionThresh value='0' /><!-- in the rage [0-255]-->
<getMarkerDetectionThresh />
<setMarkerDetectionThreshAuto value='true|false' />
<setVisionDebugMode value='true|false' />
<getVisionDebugMode />
<setVelocity value='100' /><!-- in mm/s. the rage [(-500) - (+500)] mm -->
<getVelocity />
<getAngle />
<turnCounterClockwise />
<turnCounterClockwise value='45' />
<turnClockwise />
<turnClockwise value='45' />
<moveForward />
<moveForward value='40' /><!-- distance in mm -->
<moveBackward />
<moveBackWard value='40' /><!-- distance in mm -->
<stop />
<showAxes value='true|false' />
<setCameraShift x='' y='' z='' />
<getCameraShift />
<setCurrentMap name='name' />
<setCurrentMarkerBasedCorrection value='true|false' />
<setNextMarkerBasedCorrection value='true|false' />
<setBlindPoseInterpolation value='true|false' />
<setAngleCorrectionThresh value='5' /><!-- in degree -->
<setAngleStraightThresh value='2' /><!-- in degree -->
<getAngleCorrectionThresh />
<getAngleStraightThresh />

<!-- output -->
```

```
<!-- output for getMaps command:  -->
<map name='WS-L4-19' x='0' y='0' z='0'>
      <marker name='Hiro' x='0' y='0' z='0' size='96' angle='90' />
      <marker name='Kanji' x='-1152' y='-311' z='0' size='200' angle='90' />
      <path between='Kanji' and='Hiro' />
</map>
<map name='map2' x='0' y='0' z='0'>
</map>
<map name='map3' x='0' y='0' z='0'>
</map>
<!-- etc -->

<!-- boolean output:  -->
<true />
<false />

<!-- position output:  -->
<location x='' y='' z='' /><!-- in mm -->

<!-- output for getName:  -->
<name></name>

<!-- output for getNearestMarker:  -->
<marker name='name' x='0.0' y='0.0' z='0.0' size='0.0' angle='0.0' />

<!-- output for getMarkerDetectionThresh:  -->
<markerDetectionThresh value='100' />

<!-- output for getVelocity:  -->
<velocity value='200' />

<!-- output for getAngle:  -->
<angle value='45' />

<!-- output for getAngleCorrectionThresh -->
<angleCorrectionThresh value='3' />
```

```
<!-- output for getAngleStraightThresh -->
<angleStraightThresh value='3' />
<!-- output for getVisionDebugMode -->
<visionDebugMode value='true|false'/>

<!-- output for Exceptions:  -->
<navException>Description of the exception</navException>
```

Adding a new marker

Two options are available to add new markers:

- Adding a marker from a third party library compatible with ARToolkit
- Adding a customized marker

Adding a marker from a third party library compatible with ARToolkit

In the directory **`PROJECT_DIRECTORY/patternMaker/examples/ARToolKit_Patterns`** you will find pattern files readable by ARToolKit. These patterns are used by ARToolKit to identify the markers and distinguish between them.

Let's consider that you want to add support to the marker represented by the pattern file **`4x4_100.patt`**.

1. First you will need to copy this pattern from **`PROJECT_DIRECTORY/patternMaker/examples/ARToolKit_Patterns`** directory to **`PROJECT_DIRECTORY/ARToolKit/bin/Data/`**.
2. Then you will have to edit the file **`PROJECT_DIRECTORY/ARToolKit/bin/Data/object_data_2`**
    a. Increment the number of markers (line 2)
    b. add the following lines at the end of the file
    ```
    4x4_100      # unique name of the marker (to use in the marker maps)
    Data/4x4_100.patt   # path to the pattern file
    194.0        # real size of the printed pattern (in mm)
    0.0 0.0      # coordinates of the center (always set to 0 0)
    ```
3. Then go to **`PROJECT_DIRECTORY/patternMaker/examples/gif`** and print the file **`4x4_384_100.gif`**

The new marker is now recognisable by the software. The last tasks to perform are the placing

of the printed marker at a given position and orientation on the ceiling, and the modification of the marker map accordingly.

Modifying the marker map (XML description) consists in adding a **<marker />** tag and as many **<path />** tags as there are links (possible direct paths) between the added marker and the existing ones.

The marker tag looks like below:

```
<marker name='4x4_100' x='200' y='10' z='0' size='194' angle='90' />
```

The name should be the same as in step 2 (**4x4_100**), the coordinates (in mm) are relative to the map.

If we consider there are two direct paths between the added marker and two existing markers let us say marker X and Y, the additional **<path />** tags should look like below:

```
<path between='4x4_100' and='X' />
<path between='4x4_100' and='Y' />
```

The whole XML description is summed up hereafter:

```
<setMaps>
      <map name='WS-L4-19' x='0' y='0' z='0'>
            <!-- coords, size in mm, angle in deg -->
            <!-- markers -->
            <marker name='X' x='0' y='0' z='0' size='96' angle='90' />
            <marker name='Y' x='-1152' y='-311' z='0' size='200' angle='90' />
            <marker name='4x4_100' x='200' y='300' z='0' size='194' angle='90'
/>
            <!-- links between markers -->
            <path between='4x4_100' and='X' />
            <path between='4x4_100' and='Y' />
      </map>
</setMaps>
```

This XML description is to be provided to the XML interface of the navigation API whether as a string (using **`string readXml(string xmlString)`** method) or as an XML file (using **`string readXmlFile(string pathToFile)`** method).

Adding a customized marker

In order to add your own customized marker, you will need to train the software so that it can recognize it. The training phase consist in showing the printed pattern to the camera while running a specific program provided by ARToolkit library. This program will create a pattern file which holds the characteristics of the customized marker. Once this pattern file is created, the steps are the same as in the previous section (Adding a marker from a third party library compatible with ARToolkit).

To know how to perform the marker training with ARToolkit, please visit the following link: http://www.hitl.washington.edu/artoolkit/documentation/devmulti.htm#otherpatterns