SentiBotics Development Kit Trial
Quick Start Tutorial
2016-02-29

Neurotechnology
1 Introduction

SentiBotics is robotics development kit with reference mobile manipulator intended for robotics researchers and developers, who need a "ready-to-go" platform for algorithm development. The kit consists of mobile robot, and accompanying ROS(Robot Operating System)-based software with full algorithm source code. For more information about SentiBotics see Neurotechnology website.

SentiBotics trial package allows to try SentiBotics kit and explore its capabilities. It includes a set of SentiBotics software components (excluding source code, algorithm descriptions, and physical robot itself), allowing to simulate SentiBotics robot in popular Gazebo simulator. Trial package license is valid for 30 days, and requires persistent Internet connection. We recommend to run SentiBotics trial on machine with i7 CPU, and at least 8 GB of RAM.

2 Configuring computer

This document provides step-by-step instructions how to start and use SentiBotics trial kit. Install Linux Ubuntu 14.04 operating system, and ros-indigo-desktop-full package from APT repositories.

2.1 Installing SentiBotics trial

Provided set of scripts installs additional software dependencies, sets up environment variables, installs SentiBotics trial packages and associated data:

```
./setup.sh
```

After completing installation we recommend to watch video tutorials (see Sec. 3.4), which show how to run demos and test robots capabilities.

3 Simulation enviroment and robot control

This section describes how to control the robot with control pad or text commands.

3.1 Run simulator

Start Gazebo simulator and Rviz visualization tool:

```
roslaunch sentibotics_tutorials gazebo_with_algorithms.launch
gazebo_gui:=true rviz:=true.
```

Gazebo simulator GUI Fig. 1 and Rviz Fig. 2 should appear. Press play button in Gazebo GUI, to begin simulation.

For slower machines it may be more convenient to exclude Gazebo GUI/Rviz. It can be done by setting corresponding arguments to false. If Gazebo GUI is not used, simulation can be started by running start_simulation script from sentibotics_tutorials.

---

1Current version is tested and developed with ROS Indigo and Ubuntu 14.04.
2http://www.gazebosim.org
3see http://wiki.ros.org/indigo/Installation/Ubuntu
Figure 1: SentiBotics robot in Gazebo simulator.

Figure 2: Rviz visualization tool.
3.2 Control using gamepad
Plug the gamepad into USB 2.0 port of your computer and start joystick node `rosrunc joy joy_node`. You may test your gamepad with `jstest` tool, which can be installed from APT repository. When robot is started platform movement controls are in ON state, and arm movement controls are in OFF state. START button is used for switching between platform and arm control. Robot status can be viewed by: `rostopic echo /sentibotics_platform/status`

- For robot platform controls see Fig. 3
- For robot arm controls see Fig. 4

3.3 Using SentiBotics functionality
SentiBotics software includes SLAM and autonomous navigation, object learning and recognition, and autonomous object manipulation functionality. This section provides an examples how to use it.

**Behaviour node.** Behaviour node is main SentiBotics functionality testing and demonstration tool. It is python node, functioning as a glue between lower level modules (e.g. mapping, navigation, object recognition and manipulation, etc.). It provides an easy interface for control aforementioned functionalities and serves as a framework for implementing higher level robot behaviour, like object delivery (which uses navigation, object recognition and object grasping). Currently `behaviour` contains 4 basis classes (arm control, experience map, platform control, and object recognition). Those classes have custom access to robot control and main class `behaviour.py` cover more global tasks like grasp, find or put.

In order to start behaviour node, open another terminal and type `rosrunc sentibotics_behaviour sentibotics_behaviour_node`.

**Shell scripts and programming samples.** Package `sentibotics_tutorials` contains a set of auxiliary shell scripts and C++ programming samples, which allow to use SentiBotics functionality (see `sentibotics_tutorials/src` and `sentibotics_tutorials/scripts`).

3.3.1 SLAM and autonomous navigation
First, map the environment by manually driving the robot around with control pad or commands, and ensure the map on Rviz looks to be correct(Select InteractiveMarkers to visualize experiences). Note, that mapping is performed automatically when the robot is driven by the user. In order to test autonomous navigation, we recommend to start with simple maps (e.g. two meters forward, 45 degree turn, and another two meters forward) and proceed to more complex ones, e.g. with visual loop closing as in Fig. 5.

- Driving platform directly (without using map) with given commands:
  ```
  self.platform.execute_command(delta_x, delta_theta)
  e.g. self.platform.execute_command(1, 0)
  ```

  This command don’t require a map.
Platform controls

Cancel platform movement and turn off platform controls.
Pause/resume platform movement.
Turn on/off platform movement controls.
Switch controls between platform/arm.

Platform movement:
Up - move forward.
Down - move backward.
Left - turn left.
Right - turn right.

Increase platform speed.
Decrease platform speed.
Hold to move platform at decreased speed.
Hold to move platform at increased speed.

Figure 3: Mobile platform controls

- Driving platform to a given experience:
  
  ```python
  self.travel_to_experience(experience_id)
  
  e.g. self.travel_to_experience(0)
  
  to navigate robot to initial experience
  ```
  
  This command requires a map.

- Driving platform to a pose \(x, y, \theta\) (in `/odom` coordinates). This command will drive the robot to experience, closest to given target pose, and then drive directly to it:
  
  ```python
  self.travel_to_pose(Pose2D(x, y, $\theta$))
  ```
Arm controls

Select shoulder_pan_joint.

Execute arm park action.

Select shoulder_pitch_joint.

Select elbow_joint.

Select elbow1_joint.

Up/Down - move selected joint.

Turn on/off joint movement controls.

Switch controls between platform/arm.

Select wrist_rotate_joint.

Select gripper_joint.

Select wrist_joint.

Figure 4: Robotic arm controls

e.g. self.travel_to_pose(Pose2D(1.0, 0.0, pi/4))

This command also requires a map.

Also see sentibotics_tutorials/src/autonomous_navigation_tutorial.cpp for an example how to move robotic platform and call navigation functionality. You may test it by roslaunch sentibotics_tutorials autonomous_navigation.launch.
3.3.2 Object learning and recognition

Object learning. Place an object in front of the robot in short range camera field of view. Make sure that the object is not obscured by the environment or robot. There shouldn’t be any other kind of object in the short range camera field of view to avoid misclassification. The same object should be learned in different poses for better object recognition. After learning is complete, we recommend to remove delete incorrectly learned object instances and test recognition by placing learned object in various positions and orientations, until acceptable recognition quality is achieved.

- To switch on object learning:
  ```python
  self.object_recognition.learn_object("object_name")
  ```
  or execute script `roslaunch sentibotics_tutorials learn_object object_name`

- To stop learning:
  ```python
  self.object_recognition.stop_learning()
  ```
  or execute script `roslaunch sentibotics_tutorials stop_learning`

- Switching object recognition on and off:
  ```python
  self.object_recognition.recognition_on()
  self.object_recognition.recognition_off()
  ```
  or execute corresponding scripts `roslaunch sentibotics_tutorials recognition_on` and `roslaunch sentibotics_tutorials recognition_off`

- Start object recognition mode, result is any object detected return after `RECOGNITION_TRIES` (default: 10) times. This command also associates
recognized objects to experiences, so the robot knows where they are located.

```python
self.recognize_objects()
```

- Deleting incorrectly learned object instances:
  - delete all objects matched
    ```python
    self.object_recognition.delete_all_matches()
    ```
  - delete all objects matched as object name
    ```python
    self.object_recognition.delete_matches("object_label")
    ```
  - delete all objects not matched as object name
    ```python
    self.object_recognition.delete_not_matches("object_label")
    ```
  - stop deleting
    ```python
    self.object_recognition.stop_deleting_matches()
    ```

Also see `sentibotics_tutorials/src/object_learning_tutorial.cpp` for an example how to call object learning and recognition from your software. You may test it by `roslaunch sentibotics_tutorials object_learning_and_recognition.launch`.

### 3.3.3 Object grasping and manipulation

We recommend to start testing with simple grasping scene: single learned and graspable object on plain floor. After object is learned and successfully recognized, edit `ROS_HOME/sentibotics_arm_vision/objects/object_name/params.yaml` and set `graspable` parameter to 1, also set corresponding ROS parameter to 1, or restart session.

- Parking robotic arm:
  ```python
  self.park_arm()
  ```

- Autonomous grasping of learned object. The robot will try to recognize given object and grasp it with its manipulator. If recognized object is too far, the robot tries to move closer, until grasping is possible. If no recognition occurs, the robot tries looks around current pose.
  ```python
  self.grasp_object("object_label")
  ```

- Placing grasped object into a box:
  ```python
  self.place_object()
  ```
• Autonomous object delivery. The robot will drive to experience, associated with given object, and try to recognize it directly. If recognition occurs, the robot tries to grasp the object, put it into the box and return to an experience, where delivery command was given.

    self.bring_object("object_label")

You may test it by roslaunch sentibotics_tutorials object_grasping.launch.

3.4 Video tutorials

• Mapping and autonomous navigation
• Object learning and recognition
• Autonomous object delivery
• Autonomous object grasping
• Control the robot using ’behavior’ node

4 SentiBotics structure

This section contains information about SentiBotics software.

4.1 SentiBotics ROS package structure

4.1.1 SLAM and autonomous navigation

• sentibotics_place_recognition - visual place learning and recognition.
• sentibotics_pose_estimation - robot local pose hypotheses tracking.
• sentibotics_experience_map - SLAM and autonomous navigation.

4.1.2 Object recognition, grasping and manipulation

• sentibotics_arm - robotic arm control software, auxiliary libraries for more convenient kinematics calculations and trajectory processing.
• sentibotics_arm_vision - contains 3D object recognition, grasping pose estimation, and robot self filtering.
• sentibotics_moveit - package contains moveit trajectory planner customized for Sentibotics robot, octomap filtering libraries.
• sentibotics_moveit_config - configuration of sentibotics_moveit.

4.1.3 Message packages

• sentibotics_arm_msgs - messages for operations with robotic arm.
• sentibotics_navigation_msgs - navigation messages.
• sentibotics_dynamixel_msgs - messages for Dynamixel servo motors.
4.1.4 Higher level behavior

- sentibotics_behavior - module for implementing higher level behavior and demonstration of robot capabilities.

4.1.5 Robot description

- sentibotics_description - URDF description of SentiBotics robot.

4.1.6 Robot simulation

- sentibotics_gazebo - SentiBotics Gazebo simulator package.

4.1.7 Other

- sentibotics_common - common procedures.
- sentibotics_tutorials - tutorials and scripts.