LAR

- Ferenc

Ferenc

TurtleBot package

The TurtleBot package provides means to interact with the robot. The package is located in the `src` directory of this repository and in installed on all Turtlebots.

TurtleBot class

Initialization of the turtlebot:

```
from robolab_turtlebot import Turtlebot
turtle = Turtlebot()
```

`Turtlebot(rgb=False, depth=False, pc=False)`

Initializes Turtlebot object, the rgb, depth and pc arguments determine whether rgb images, depth maps and point clouds will be downloaded from the camera.

`cmd_velocity(linear=0, angular=0) -> None`

Commands velocity (m/s, rad/s) to the robot, in order to maintain the velocity this command must be called with rate of 10 Hz.

`get_odometry() -> [x,y,a]`

Returns odometry, i.e. estimation of the current position of the Turtlebot, this position is references to the boot up position or to the last call of the `reset_odometry`. `[x,y]` are Cartesian coordinates of Turtlebot in meters and `[a]` is the angle in Radians.

`reset_odometry() -> None`

Resets the odometry to `[0,0,0]` i.e. sets new starting position.

`get_rgb_image() -> image`

Returns RGB image as 480x640x3 Numpy array. In order to use RGB images initialize Turtlebot class with rgb stream i.e. `Turtlebot(rgb=True)`,

`get_depth_image() -> image`

Returns depth image as a 480x640x1 Numpy array, the depth is in meters. In order to use depth images initialize Turtlebot class with depth stream i.e. `Turtlebot(depth=True)`.

`get_point_cloud() -> point_cloud`

Returns point cloud in form of a 480x640x3 Numpy array. Values are in meters and in the coordinate system of the camera. In order to use point cloud initialize Turtlebot class with pc stream i.e. `Turtlebot(pc=True)`.

`play_sound(sound_id=0) -> None`

Plays one of predefined sounds: (0 - turn on, 1 - turn off, 2 - recharge start, 3 - press button, 4 - error sound, 5 - start cleaning, 6 - cleaning end).

`register_button_event_cb(fun) -> None`

Register callback for the button event, when the button is pressed the fun is called. The `fun` callback should have form of a function with one argument `button_cb(event)`, the `event` object has two variables `event.button` which stores the number of button that producing this event and `event.state` which stores the event type (0:released, 1:pressed). The `event` object is actually ROS message kobuki_msgs/ButtonEvent.

`register_bumper_event_cb(fun) -> None`

Register callback for the bumper event, when the robot hits something the fun is called. The `fun` callback should have form of a function with one argument `bumper_cb(event)`, the `event` object has two variables `event.bumper` which stores the bumper number (0:left, 1:center, 2:right) and `event.state` which stores the event type (0:released, 1:pressed). The `event` object is actually ROS message kobuki_msgs/BumperEvent. Do not hit anything!

`is_shutting_down(self) -> bool`

Returns true if CTRL+C was pressed. Use in the main loop as: 'while not turtle.is_shutting_down():'

`get_rgb_K(self) -> K`

Return K matrix (3x3 numpy array) for RGB camera.

`get_depth_K(self) -> K`

Return K matrix (3x3 numpy array) for depth camera.

Marker Detector

`detect_markers(image) -> detections`

Detects markers in a image and returns detections as a list of tuples `(corners, id)`, where `corners` is list of points 4x2, and `id` is the identification number of detected marker.

`show_markers(image, detections)`

Draw detection into the image. See `show_marker.py` as an example.

Tools

Some tools you might find helpful

keyop

In order to control the robot movements from command line, run the following command:

```
roslaunch kobuki_keyop safe_keyop.launch
```

Examples

Examples are stored in `scripts` in this repository.

The show_depth.py demonstrates the point cloud acquisition and visualization.

The show_markers.py demonstrates the acquisition of rgb images and detection of checkpoint markers.

The random_walk.py demonstrate the use of depth sensor to avoid obstacles. The robot moves in straight line until it detects obstacle in the point cloud. Then it rotates until the obstacle disappear.

- bumper_test.py shows how to use `register_bumper_event_cb`, the `register_button_event_cb` is used in a similar way.
- example_move_1m.py
- test_sensors.py

Robotic Operating System

The software environment where the TurtleBot and sensors live is provided by Robotic Operating System (ROS). We heve made our best to hide the ROS part of the system from you so you can focus on the task. However, if interested you can found more information about ROS at www.ros.org.

Useful stuff

- In order to save array (depth image, points) one can use numpy's save function.

```
from robolab_turtlebot import Turtlebot
impot numpy as np

turtle = Turtlebot()
depth = turtle.get_depth_image()

np.save('depth.npy', depth)

# -----

depth = np.load('depth.npy')
```

Resources

- FAQ
- Presentation
- ROS: Robotic Operating System
- ROS Wiki