Institute for Cognitive Systems
Technische Universität München
Prof. Gordon Cheng

Humanoid Robotics Systems
Tutorial 7: Walking, Talking, Blinking

Before you begin

- In this tutorial, you will use ROS to access a set of native high level functionalities of NAO robot. These functionalities are not initialized in the basic `nao_bringup` ROS interface. Therefore you will have to call them manually. This can be done using the following command: "`roslaunch nao_apps nao_<module>.launch`", where `<module>` is the name of the interface you call (e.g. `nao_leds.launch`). Take a look at `https://wiki.ros.org/nao_apps` for more details. If you are curious and want to see the Pyhton code of the different nodes, feel free to take a look at `/opt/ros/indigo/lib/nao_apps`. The corresponding launch files are located in `/opt/ros/indigo/share/nao_apps/launch`.
- Review the steps required to create a new ROS action client in section 3.4 of the "ROS survival guide". Alternatively, you can check the following tutorial on the official ROS website: `https://wiki.ros.org/actionlib_tutorials/Tutorials/`
- Information about topics, services and actions of the NAO interface can be found at the following address: `http://wiki.ros.org/nao_apps`.

Exercise 1: NAO, in the blink of an eye

In this first exercise you will have to make NAO's eyes blink based on bumper interrupts:

- Each time the left foot bumper is pressed, the LEDs should blink with red color for 2 seconds.
- Each time the right foot bumper is pressed the LEDs should blink with green color for 4 seconds.

There are two ways to proceed (just select one):

1. the elegant way consists in coding a ROS action client for the "`/blink`" action server (bridging ROS to NAOqi). A new action goal should be set whenever a bumper event is caught by a dedicated callback.
2. the dirty (but working) way consists in using classic publishers to access the action server functionalities. A new message containing the action goal should be published in "`/blink/goal`" whenever a bumper event is caught by a dedicated callback.

Hints:

- The bumper states are published in the "`/bumper`" topic. Do not forget to run "`roslaunch nao_apps tactile.launchh`" in a separated terminal to have this topic published on the ROS network.

- In case you select the action client way, the message structure is defined in the file `/action/Blink.action` of your workspace.
- In the other case, the message published in `/blink/goal` shouyld be of type: "`naoqi_bridge_msgs::BlinkActionGoal`". The important parameters of the message are[1]:
  - `msg.goal_id.id` : this is a unique ID for the requested action. This is a `string` variable so you should initialize it with a unique string.
  - `msg.goal.colors` : this is a vector of colors in `std_msgs::ColorRGBA` format. NAO will choose randomly a color from this vector for blinking.
  - `msg.goal.blink_duration` : this variable defines the duration of the blink in seconds.
  - `msg.goal.blink_rate_mean`: this variable defines the mean of the blinking rate.
  - `msg.goal.blink_rate_sd`: this variable defines the standard deviation of the blinking rate.
  - `msg.goal.bg_color` : this variable defines the color (in "`std_msgs::ColorRGBA`" format) when the robot is not blinking. This variable can be left empty.

  Keep in mind that NAO will continue blinking until a new message of type: "`naoqi_bridge_msgs::BlinkActionGoal`" is published, or until the current action is canceled by publishing a dedicated message into the topic: "`/blink/cancel`"

Exercise 2: Let's talk NAO

In this second exercise you will have to implement a module for NAO speech generation and recognition:

- When the front tactile button is pressed NAO should start speech recognition and memorize all recognized words.
- When the middle tactile button is pressed NAO should stop speech recognition and say all memorized words as one sentence.

Hints:
Before you start speech recognition you should create a vocabulary which is a list of words NAO should recognize. If you feel confident with the ROS actionlib, you can code a client for the corresponding action server. Otherwise, use "`voc_params_pub`" to publish vocabulary to the topic: "`/speech_vocabulary_action/goal`". The format of the message is "`naoqi_bridge_msgs::SetSpeechVocabularyActionGoal`" and the important parameters are:

- `msg.goal_id.id` : this is a unique ID for the requested action. This is a `string` variable so you should initialize it with a unique string.
- `msg.goal.words` : this is a vector of `string` which defines the vocabulary.

After you defined the vocabulary, you can start speech recognition by calling service client "`recog_start_srv`" with empty request. To stop recognition use "`recog_stop_srv`". For talking you should use "`speech_pub`" which publishes the desired text to the NAO text to speech module. The topic name is "`/speech_action/goal`", and the message format is: "`naoqi_bridge_msgs::SpeechWithFeedbackActionGoal`". The important parameters of this message are:

[1] Note that the exact action message definition can be found in `/opt/ros/indigo/share/naoqi_bridge_msgs/action/Blink.action`

- `msg.goal_id.id` : this is a unique ID for the requested action. This is a `string` variable so you should initialize it with a unique string.
- `msg.goal.say` : this is a `string` variable which defines the speech text.

Note: If NAO stands up during speech recognition, it will execute a set of undesired motions that may interfere with other control tasks and generate posture instabilities. You can deactivate this set of behaviors by simply pressing twice on the torso button.

Exercise 3: Take a walk with NAO

In this last exercise you need to implement simple walking on NAO:

- Implement walking along the straight line.
- Implement turning to the left and to the right.

Hints:
Walking is here achieved using the Aldebaran's controller. To get NAO to walk, you just need to publish the desired position using "`walk_pub`" to the topic "`/cmd_pose`" (no actionlib here). The internal controller will then figure out the number of steps to be executed by NAO and will execute them. The format of the message is "`geometry_msgs::Pose2D`" and the important parameters are:

- "`msg.x`" : this variable defines the x coordinate of the desired position.
- "`msg.y`" : this variable defines the y coordinate of the desired position.
- "`msg.theta`" : this variable defines the desired orientation.

First, you should implement the function "`stopWalk`" which stops the movement whenever it is called. To stop walking you can call the service "`/stop_walk_srv`" with empty request. Then, use your `stopWalk` function to stop walking when the feet no longer contact the ground or when an event is caught in the rear tactile button. Information about current foot contact is published in the topic: "`/foot_contact`" and you can use the callback function "`footContactCB`" to get the data. Next, implement function "`walker`" which should initialize motion towards desired position. For testing, use rear tactile button to start walking. When the button is pressed the robot should start moving, if the contact with the ground is lost or if the rear tactile button is pressed a second time, it should immediately stop.

Submission:

- All your data must be in a `.zip` file.
- Name this file "`T7_group_<letter>.tar.gz`" , where `<letter>` is a placeholder for your group name, either A, B, C, D, or E.
- Submit the `.tar.gz` file of your group to `quentin.leboutet@tum.de`.
- Please don't forget to add the TUM identifiers of all group members to the email!
- Deadline: **19.12.2018, 23:59**