Assignment 5

To reach a goal, like moving its base along a trajectory to a new position, or moving its arm to a new pose, the robot needs quite some time. Often the robot has to pursue multiple goals in parallel. The purpose of this assignment is to get you acquainted with *actionlib*. This ROS library allows you to write a node that can manage one or more long-running actions implemented by other nodes.

# Learning goals

This assignment is specifically targeted at following learning goal:

* use the ‘actionlib’ library for controlling long-running actions

and extends your knowledge of some learning goals you have already dealt with in previous assignments:

* use command line tools to monitor and debug a running ROS application
* analyse a ‘Computation Graph’, comprising ‘nodes’, ‘topics’, ‘messages’, ‘services’, ‘parameters’, and a ‘ROS Master’
* build new ROS packages and use existing ones
* write launch files to start a collection of nodes and to upload parameters
* write new ROS nodes using the C++ (or Python) programming language
* write a node to drive a real or simulated mobile robot using ‘Twist’ messages

# Getting ready

Assumption is that you already know how messages, services, spinning and callbacks work. If not you should head back to the ROS Tutorials (<http://wiki.ros.org/ROS/Tutorials>) and specifically study the following tutorials:

* Creating a ROS msg and srv
* Writing a Simple Publisher and Subscriber
* Writing a Simple Service and Client

You are also advised to (re)read following chapters from the book “A Gentle Introduction to ROS” (<https://cse.sc.edu/~jokane/agitr/>):

* Chapter 3. Writing ROS programs
* Chapter 8. Services

Then dive into *actionlib* (<http://wiki.ros.org/actionlib>) and the associated actionlib tutorials (<http://wiki.ros.org/actionlib/Tutorials>). Wat functionality does it provide? How to use it?

In this and following assignments we will be using the Turlebot robot. Get to know this robot:

* TurtleBot.com (<http://www.turtlebot.com/>)
* TurtleBot on the ROS wiki (<http://wiki.ros.org/Robots/TurtleBot> )
* Learn TurtleBot and ROS (<http://learn.turtlebot.com/>)

Then, before you start programming, study the *turtlebot\_actions* package (<http://wiki.ros.org/turtlebot_actions>) which provides several basic actionlib actions for the TurtleBot. In particular look at the source code of the *turtlebot\_move\_action\_server* (which you can find under turtlebot\_actions‘’ in this repository: <https://github.com/turtlebot/turtlebot_apps> )

Finally install the binary packages you will need in this assignment:

* sudo apt-get install ros-indigo-turtlebot
* sudo apt-get install ros-indigo-turtlebot-apps
* sudo apt-get install ros-indigo-turtlebot-rviz-launchers
* sudo apt-get install ros-indigo-turtlebot-simulator

# The Tasks

You are going to develop a robot node that uses the actionlib to make a robot do a long running action, in this case to make a drive along a triangular path. First you will test it using a simulated robot and then you are asked to make it work using a real Turtlebot robot.

## Create an actionlib client to control a long-running move action

Your first task is to write a ROS node: *move\_triangle,* that functions as a client to the *turtlebot\_move\_action\_server* in order to implement following requirements:

* The node must subscribe to a topic called */cmd*. On this topic it must be able to receive *Triangle* messages, defined as follows:  
   *float32 sideLength  
   bool cw*
* When receiving a *Triangle* message the node must make the robot move along a triangle with the given side length).The bool cw should make the turtle drive clockwise (cw == true) or counterclockwise (cw == false).
* The node must communicate with the *turtlebot\_move\_action\_server* node via the actionlib and must use TurtleMove.action.
* The node must print the feedback it gets from the *turtlebot\_move\_action\_server.*
* Of course you must provide a proper launch file to start all required nodes.

/cmd ccmdcmd\_square

Twist

tf

TurtleMove.action

For testing use the same setting as in previous assignments: Stage simulator with empty.world.

You may also choose to use Gazebo if you have sufficient processing power and good GPU. In this case have a look at following page on the ROS Wiki: <http://wiki.ros.org/turtlebot_gazebo/Tutorials/indigo/Gazebo%20Bringup%20Guide>

Hints:

* Remember the first assignment of this course. You may recognize the same requirements. Maybe you can reuse some of your own code, or maybe you decide to completely start from scratch now that you have more experience. Your choice!
* The *TurtleMove.action* is already defined in the *turtlebot\_actions* package. You do not have to (re)define this one. Just use it.
* In a launch file you can give a node its own terminal screen using a *launch-prefix* attribute:  
    
  <node name="foo" pkg="bar" type="bar\_node" output="screen" launch-prefix="xterm -e" />

## Make it work using a real TurtleBot [OPTIONAL]

The teacher will provide a real TurtleBot. Your task is to replace the Stage simulator by the real TurtleBot. Both *move\_triangle* and *turtlebot\_move\_action\_server* nodes must run on your laptop.

Hints:

* The ROS Master is already running on the TurtleBot. What does this mean?
* You have to make a new launch file. Analyse what the TurtleBot publishes on and subscribes to using ROS commands that should be well known to you now.

# What to Submit

Your submission for this assignment must contain following parts:

1. The source code, i.e. a zip file containing the package
2. A YouTube movie to show the TurtleBot doing what it is supposed to do.
3. A document which should:
   1. explain how everything works and include a computation graph,
   2. describe test scenarios
   3. provide one or more interesting screendump(s)