Project 2: Kinematics

Implement the project by modifying *project_2* package provided to you and changing the name to look like ***<my_uid>_project_2***. Hand in this project by uploading the package via the ELMS website.

1. Before you start

Most of the instructions here are provided for Python. Nevertheless, feel free to use C++ if you want!

In the first part of this project, you will implement models for the forward and inverse kinematics of a differential drive robot. This project will assume the following conventions:

- Poses will consist of x and y coordinates (measured in meters) and a coordinate (in radians).
- Each atomic Action will be rotating the left and right wheels at a given speed for some set period of time.
- The robot will be described by three parameters. First, the axle length will be the distance between the centers of the wheels in meters. Second, the wheel radius is the radius of the wheels. Lastly, we have a maximum speed at which the wheels can rotate in radians per second, which is only used in the inverse kinematics.

Download the code provided for project and put packages *<my_uid>_project_2* and *wheeled_robot_kinematics* into your ros workspace. Build the packages and make sure that they can be found by ROS i.e. you can *roscd* into them.

2. Pure Python / C++

The heavy mathematical lifting for calculating kinematics in this project will be done in *project_1/src/project_1/kinematics.py.* The first function for you to define is for forward kinematics:

```
def forward (p, a, rd ):
    (x, y, theta) = p
    (vl, vr, t) = a
    (axle_length, wheel_radius, max_speed) = rd
    return 0.0, 0.0, 0.0
```

Given a pose and an action (given as 3-tuples) and a robot description (3-tuple), return the new pose. **Note** that *max_speed* is not needed to compute forward kinematics.

The second function defines inverse kinematics:

```
def inverse (p0, p1, rd):
    (x0, y0, theta0) = p0
    (x1, y1, theta1) = p1
    (axle_length, wheel_radius, max_speed) = rd
    return [ ]
```

Given two poses and a robot description, return an array of actions that will move the robot from one pose to the other. In general, any pose can be reached using three discrete actions: turn, move in a straight line, turn. However, **certain poses can be reached in a single move** and there exists an approach that requires less than three moves to get to anywhere.

To test the code, you are given two python scripts. *project_2/src/forward.py* will call your forward kinematic function based on command line arguments and print out the resulting pose. *project_2/src/inverse.py* will call your inverse kinematic function based on command line arguments, print out the resulting set of actions, and use your forward kinematic model to figure out where the plan actually ends up.

3. ROS Python / C++

Using the first half of the rospy service tutorial as a guide, write a python/C++ script in *project_2/src/services.py (.cpp)* that creates services for both the forward and inverse kinematics, as dened in *wheeled_robot_kinematics/srv*.

- ROS Service tutorial: Python, C++
- ROS parameter server: link

- Your node should be named "*kinematics*".
- The two services should be called *"~forward"* and *"~inverse"*. This will make the full name of your service */kinematics/forward (inverse)*.
- Be sure to use your libraries from the first part i.e. this code should act as a ROS wrapper around your pure Python code.
- The robot description parameters should be read from the parameter server with the names */axle_length*, */wheel_radius* and */max_speed*.
- Make sure your script is executable (if using Python).
- Do not forget to modify *CMakeLists.txt* and *package.xml* to reflect the change of package name from ***project_2*** to ***<my_uid>_project_2.*** Also important for C++ users.

To test your code:
1. *catkin_make* your packages
2. Launch *roscore*
3. In one terminal, run your services script: *rosrun <my_uid>_project_2 services.py*

4. In another terminal, run the tests provided by the *kinematics* package: *rosrun wheeled_robot_kinematics kinematics_test.py* This should show you a number of tests and their results.

4. Grading

Forward method	25 points
Inverse method (3 actions)	25 points
Minimal inverse method (less than 3 actions)	20 points
Forward service	15 points
Inverse service	15 points