Carnegie Mellon University Africa

18-799 Introduction to Cognitive Robotics

Assignment 2
Robot Locomotion: Solution of the Go-to-Position Problem using the Divide and Conquer and MIMO Algorithms

Deadline: 18:00 Tuesday 2nd March 2021

Problem Definition
Implement the divide and conquer go-to-position algorithm and the MIMO go-to-position algorithm for a turtlebot in the ROS turtlesim simulator, extending them if necessary to ensure that the turtlebot achieves the goal location and orientation.

The program should read an input file `assignment2Input.txt`.

This file contains a sequence of commands, one command per line.

Each command comprises seven fields: a key string (either "`goto1`" or "`goto2`") followed by six floating point numbers.

- The first three numbers give the start pose of the turtle (x, y, theta, respectively).
- The second three numbers give the goal pose of the turtle (x, y, theta, respectively).

For each command, the turtle is positioned at the start pose and then drives to goal pose using the specified algorithm: divide and conquer for `goto1` and MIMO for `goto2`.

The current turtle pose is sensed by subscribing to the `turtle1/pose` topic.

The turtle is driven by publishing velocity commands on the `turtle1/cmd_vel` topic.

The Turtlesim environment should be reinitialized after the execution of each command, clearing any path displayed, but only after the user is prompted to continue to the next command (so that she or he can take time to inspect the path the turtlebot has taken).

Input
A sequence of lines, each comprising a command and start and goal pose values.

Output
The turtle should be teleported to the start location and then moved to the goal pose using the algorithms specified in the command.

Sample Input
```
goto1 2.0 1.0 3.14 8.0 8.0 0.0
goto2 2.0 1.0 3.14 8.0 9.0 0.0
```

Sample Output
See Figures 1 and 2 below.

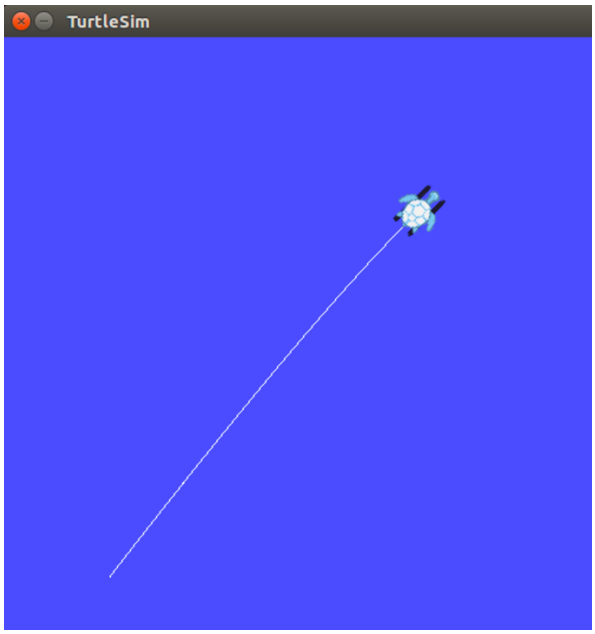

Figure 1. Divide and conquer algorithm.

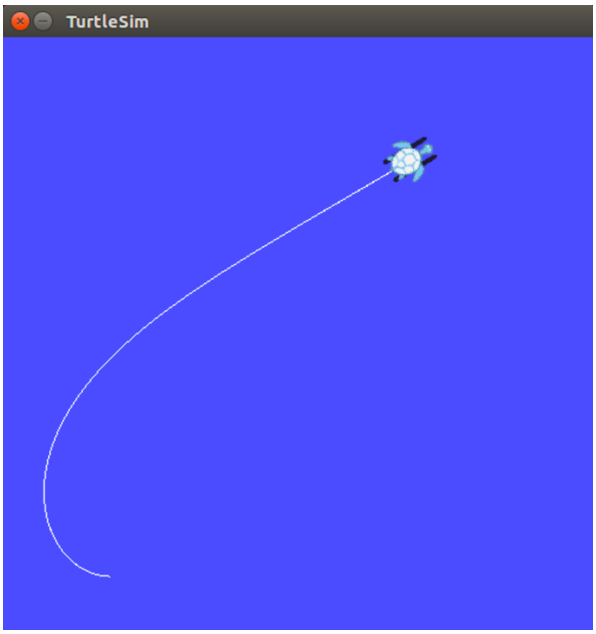

Figure 2. MIMO algorithm.

Instructions
To help you get started, a skeleton program is provided. It provides the functionality for reading the input file. Specifically, five files are provided:

```
studentid.h
studentidApplication.cpp
studentidImplementation.cpp
CmakeLists.txt
assignment2Input.txt
```

The main client code (i.e. the code that calls the functions that are required to build the application) is placed in `studentidApplication.cpp` while the functions themselves are all placed in `studentidImplementation.cpp`. The interface file `studentid.h` provides the declarations of these functions so that the client code can use them.

We adopt this approach to encourage abstract interfaces and to separate the use of functions from their implementation, thereby allowing changes to be made to the implementation without affecting the application (i.e. the client) code.

Here is how you should use these files.

First, create a package `assignment2` in the ROS workspace:

```
$cd ~/workspace/ros/src
$catkin_create_pkg assignment2 roscpp
```

(refer again to the example where we did this with for `agitr` package).

Copy the sample `CmakeLists.txt` file to `~/workspace/ros/src/assignment2/`

Created a subdirectory `~/workspace/ros/src/assignment2/data` and copy the sample input file `assignment2Input.txt` there.

Copy the sample `studentid.h` file to
`~/workspace/ros/src/assignment2/include/assignment2/`

Copy the sample `studentidApplication.cpp` file to
`~/workspace/ros/src/assignment2/src/`

Copy the sample `studentidImplementation.cpp` file to
`~/workspace/ros/src/assignment2/src/`

Build this package with `catkin_make` and run the program with `rosrun assignment2 studentid`

Verify that it works by inspecting the messages echoed to the input to the terminal.

Now, make a copy of the three source files (one `.h` and two `.cpp`) , replacing `studentid` with your student ID.

Edit each source files to customize it, replacing `studentid` with your student ID (there are additional instructions in the source files).

Edit the `CmakeLists.txt` file to customize it, replacing `studentid` with your student ID (there are additional instructions in the header of this file).

Build this package with `catkin_make` and run the program with `rosrun assignment2 <your_own_studentid>`.

Again, verify that it works by inspecting the messages echoed the input to the terminal.

Now, write the code to compute the robot position as required in the assignment. Put the target function to do this in the implementation file, the declaration in the interface file, and the function call in the application file.

Submit the three source code files named with your student ID, i.e.. `mystudentidApplication.cpp`, `mystudentidImplementation.cpp`, `mystudentid.h`, and your `assignment2Input.txt` file in a zip file named with your student ID by the deadline shown above.

The source code should contain adequate internal documentation in the form of comments. Internal documentation should include the following.

- A summary of the algorithm
- A summary of the manner in which the code was tested.

Do not forget to include your name in the internal documentation.

Marking Scheme

Marks will be awarded based on blind tests using data similar to those in the sample input above. Marks will be awarded for the effectiveness of the robot path and the time taken to achieve the goal pose, given the algorithm that is being used to effect the locomotion.

Equal marks will be awarded for each test case.