PS6: libraries and code documentation

Request:

create a library (in its own package) called "my_interesting_moves"

This library should contain mulitple member functions named mnemonically, e.g.

```
  set_goal_salute(trajectory_msgs::JointTrajectory &des_trajectory);
```

Create at least 3 interesting functions that populate trajectory messages for Baxter's right arm.

Document your API with Doxygen comments in your header file.

ALSO, include the output of "roslint" on your code. You should try to get zero complaints from roslint.

In a separate package, create a node that uses your new library to move Baxter.
It should demonstrate each of your interesting moves by populating trajectory messages
via your library's functions, and executing these using the baxter trajectory streamer action server. (see package: cwru_baxter/baxter_traj_streamer/src/traj_action_clien_pre_pose.cpp for an example action client compatible with this action server).

run:
roslaunch cwru_baxter_sim baxter_world.launch
wait for the message: "Gravity compensation was tuned off"
in another window, enable the robot with the command:

```
 rosrun baxter_tools enable_robot.py -e
```

This command will run to completion.

Start the trajectory interpolator action server:

```
 rosrun baxter_traj_streamer traj_interpolator_as
```

Leave this node running.

In another terminal, start your node. For test purposes, run the example client node:

```
 rosrun baxter_traj_streamer traj_action_client_pre_pose
```

make a movie of your resulting motions.

Include in your submission:
*the github URL's for your library and for your node (your github must include your html files for Doxygen-generated documentation)

*a zip file of your package with library, main program, and doxygen documentation

*the output of "roslint":

rosrun roslint cpplint file.cpp

*a movie of Baxter executing your designed moves

Important Note:

The gihub link for the assignment: https://github.com/TuZZiX/ROS_Robotics-my_interesting_moves

Please to and **clone it to your catkin workspace**.

I have added a new preempt response feature to traj_interpolator_as, but I cannot commit changes to cwru_baxter. However, this feature is used in the my_interesting_moves library, so you have to **replace the file**

cwru_baxter/baxter_traj_streamer/src/traj_interpolator_as.cpp

with the one in my github.

Movie of Baxter executing library example: interesing_moves_exampl_motion_demo.mp4

Doxygen generated documentation could be found at https://github.com/TuZZiX/ROS_Robotics-my_interesting_moves/tree/master/html, or under html folder. (notice that index.html is the entrance)

Screen capture of roslint and doxygen can be found under screen_capture folder.

How to run:

1. Follow important note.
2. Run following commands in your terminal:

```
cd (catkin_workspace)
```

```
catkin_make
```

```
roslaunch cwru_baxter_sim baxter_world.launch
```

```
roslaunch my_interesting_moves interesting_moves_example.launch
```

Answers:

The library my_interesting_moves contain a single file which is my_interesting_moves.h. it contains a class called Baxter_right_arm, with this class, it will be very easy to command a pose to Baxter's right arm, like:

```
Baxter_right_arm right_arm;  // create a object
right_arm.move(your_desired_pose);  // your_desired_pose is a 7x1 matrix
or a 7 elements array contain the pose you want
```

There is also a complex method but provide more freedom:

```
right_arm.add_movement(joint_pose);  // add a branch of movment
right_arm.start_move();  // start moving when you finished
right_arm.wait_for_finish(3.0); // wait for a certain time
right_arm.stop_move();  // stop it any time you want
```

stop_move function need to incorporate with modified traj_interpolator_as node.

There is also a function for checking actuator limitation which means you cannot go beyond the maximum angle of the arm hardware, to enable it, use:

```
right_arm.enable_pose_limit = true;
```

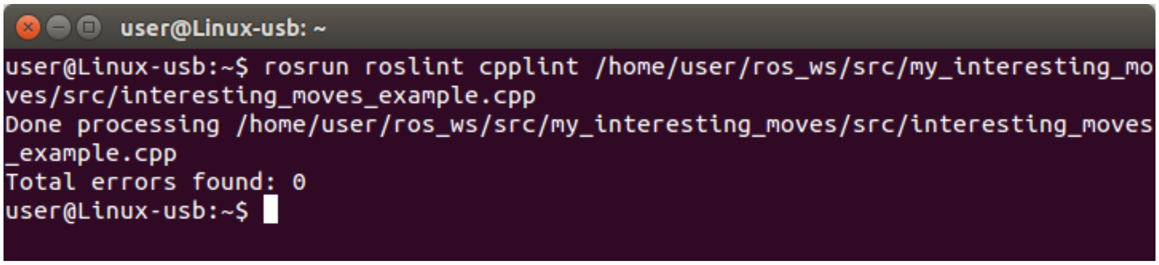
In this way, right_arm will check every pose command to it, if it is out of executable range, it will be restricting to maximum angle that robot can reach.

Also, this library includes some prefabricated motion like:

```
right_arm.move_above_table();  // prevent arm been stucked by table
right_arm.push_beer();  // push the bear off table
right_arm.wave_hand(3);  // wave hand 3 times
```

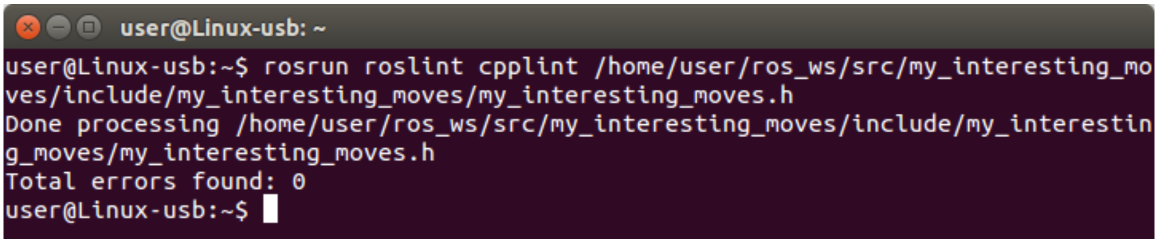
There is an example for how to use the library src/interesting_moves_example.cpp which is showed in the demo video.

The library .h file and example usage .cpp file have been regulated with the standard of roslint, *Fig.1* shows the roslint output of example usage and *Fig.2* shows the roslint output of library header file, no error in both files.



```
user@Linux-usb:~$ rosrun roslint cpplint /home/user/ros_ws/src/my_interesting_mo
ves/src/interesting_moves_example.cpp
Done processing /home/user/ros_ws/src/my_interesting_moves/src/interesting_moves
_example.cpp
Total errors found: 0
user@Linux-usb:~$ 
```

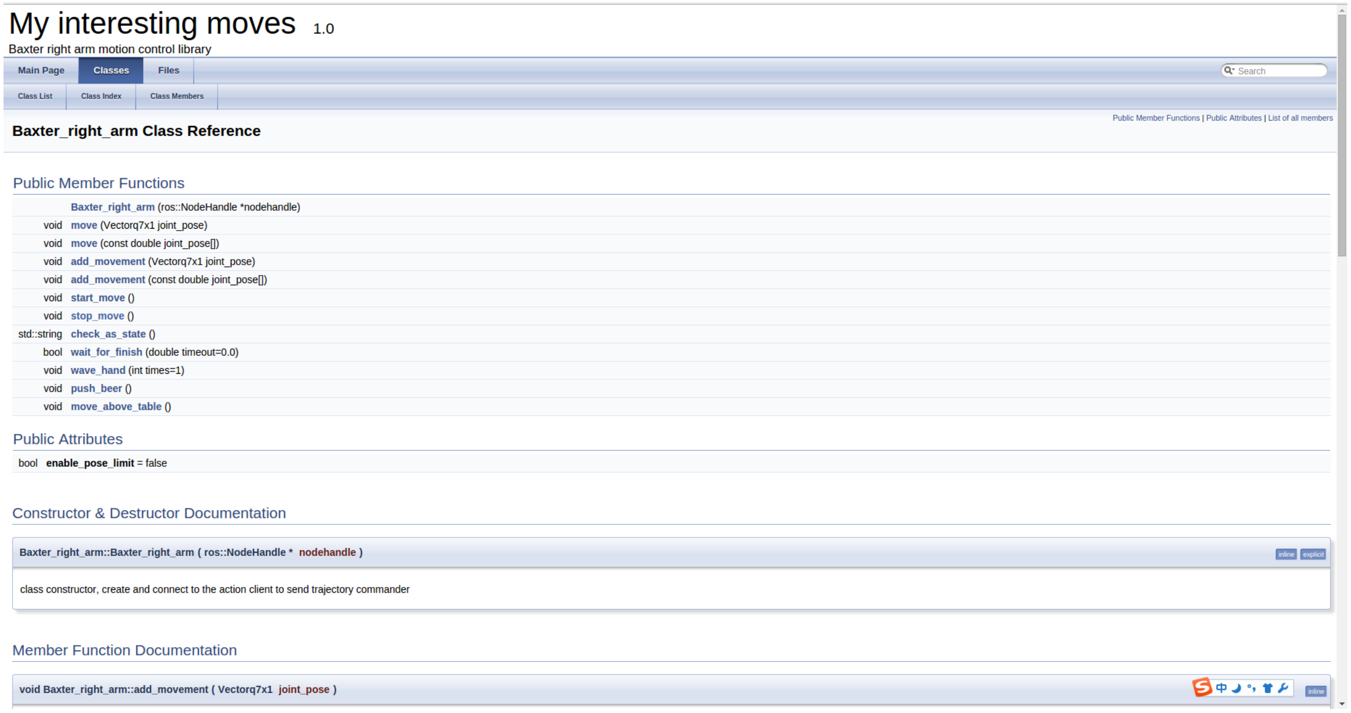
Fig.1



```
user@Linux-usb:~$ rosrun roslint cpplint /home/user/ros_ws/src/my_interesting_mo
ves/include/my_interesting_moves/my_interesting_moves.h
Done processing /home/user/ros_ws/src/my_interesting_moves/include/my_interestin
g_moves/my_interesting_moves.h
Total errors found: 0
user@Linux-usb:~$
```

Fig.2

Fig.3 and Fig.4 shows one page of the Doxygen generated documentation, and all the codes are in a good format.



My interesting moves 1.0

Baxter right arm motion control library

Baxter_right_arm Class Reference

Public Member Functions

	Baxter_right_arm (ros::NodeHandle *nodehandle)
void	move (Vectorq7x1 joint_pose)
void	move (const double joint_pose[])
void	add_movement (Vectorq7x1 joint_pose)
void	add_movement (const double joint_pose[])
void	start_move ()
void	stop_move ()
std::string	check_as_state ()
bool	wait_for_finish (double timeout=0.0)
void	wave_hand (int times=1)
void	push_beer ()
void	move_above_table ()

Public Attributes

bool	**enable_pose_limit** = false

Constructor & Destructor Documentation

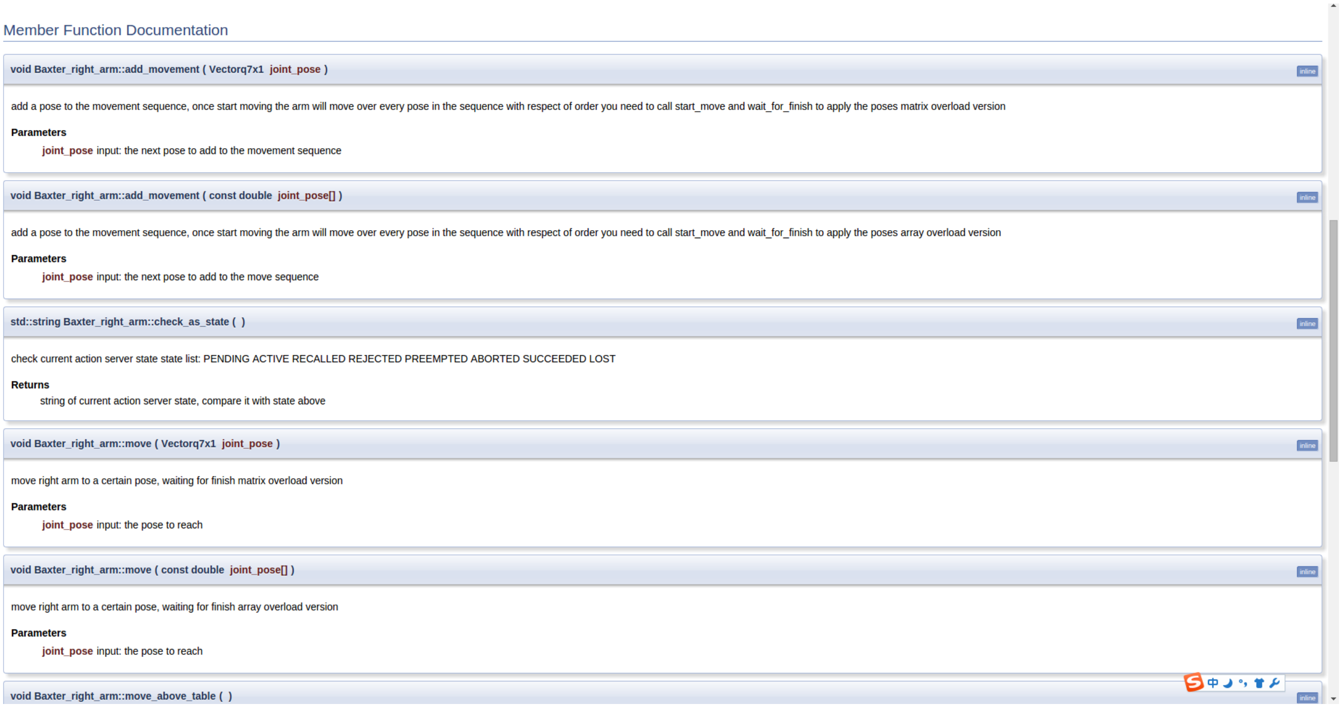
Baxter_right_arm::Baxter_right_arm (ros::NodeHandle * nodehandle)

class constructor, create and connect to the action client to send trajectory commander

Member Function Documentation

void Baxter_right_arm::add_movement (Vectorq7x1 joint_pose)

Fig.3



Member Function Documentation

void Baxter_right_arm::add_movement (Vectorq7x1 joint_pose)

add a pose to the movement sequence, once start moving the arm will move over every pose in the sequence with respect of order you need to call start_move and wait_for_finish to apply the poses matrix overload version

Parameters

joint_pose input: the next pose to add to the movement sequence

void Baxter_right_arm::add_movement (const double joint_pose[])

add a pose to the movement sequence, once start moving the arm will move over every pose in the sequence with respect of order you need to call start_move and wait_for_finish to apply the poses array overload version

Parameters

joint_pose input: the next pose to add to the move sequence

std::string Baxter_right_arm::check_as_state ( )

check current action server state state list: PENDING ACTIVE RECALLED REJECTED PREEMPTED ABORTED SUCCEEDED LOST

Returns

string of current action server state, compare it with state above

void Baxter_right_arm::move (Vectorq7x1 joint_pose)

move right arm to a certain pose, waiting for finish matrix overload version

Parameters

joint_pose input: the pose to reach

void Baxter_right_arm::move (const double joint_pose[])

move right arm to a certain pose, waiting for finish array overload version

Parameters

joint_pose input: the pose to reach

void Baxter_right_arm::move_above_table ( )

Fig.4