EECS 476 Mobile Robotics
PS 4

Peng Xu
pxx37@case.edu

1. Main idea

1) action sever & client communication

This line in client end,

```
bool server_exists = action_client.waitForServer(ros::Duration(1000.0));
```

and the constructive function of the server,

```
MobotActionServer::MobotActionServer() :
  as_(nh_, "mobot_action", boost::bind(&MobotActionServer::executeCB, this, _1),false)
// in the above initialization, we name the server "mobot_action"
//  clients will need to refer to this name to connect with this server
{
   ROS_INFO("in constructor of MobotActionServer...");
   // do any other desired initializations here...specific to your implementation
            do_inits(nh_);
   as_.start(); //start the server running
}
```

together establish an action server and client communication.

2) transmitting messages

```
   ros::Subscriber alarm_subscriber = n.subscribe("lidar_alarm",1,alarmCallback);

   ros::Rate loop_timer(100);
   while(!g_lidar_alarm) {
      goal.input = 1;

      goal.distance.resize(5);
      goal.distance[0] = 4;
      goal.distance[1] = 2;
      goal.distance[2] = 4;
      goal.distance[3] = 2;
      goal.distance[4] = 5;

      goal.angle.resize(5);
      goal.angle[0] = 0;
      goal.angle[1] = PI/2;
      goal.angle[2] = PI/2;
      goal.angle[3] = -PI/2;
      goal.angle[4] = -PI/2;
```

```
        //action_client.sendGoal(goal); // simple example--send goal, but do not specify
callbacks
        //action_client.sendGoal(goal,&doneCb); // we could also name additional callback
functions here, if desired
        action_client.sendGoal(goal, &doneCb, &activeCb, &feedbackCb); //e.g., like this

        ros::spinOnce();
        loop_timer.sleep();
    }
```

Here I use a vector to define a S shape path for the robot.

3) server end sending commands to the robot “mobot”

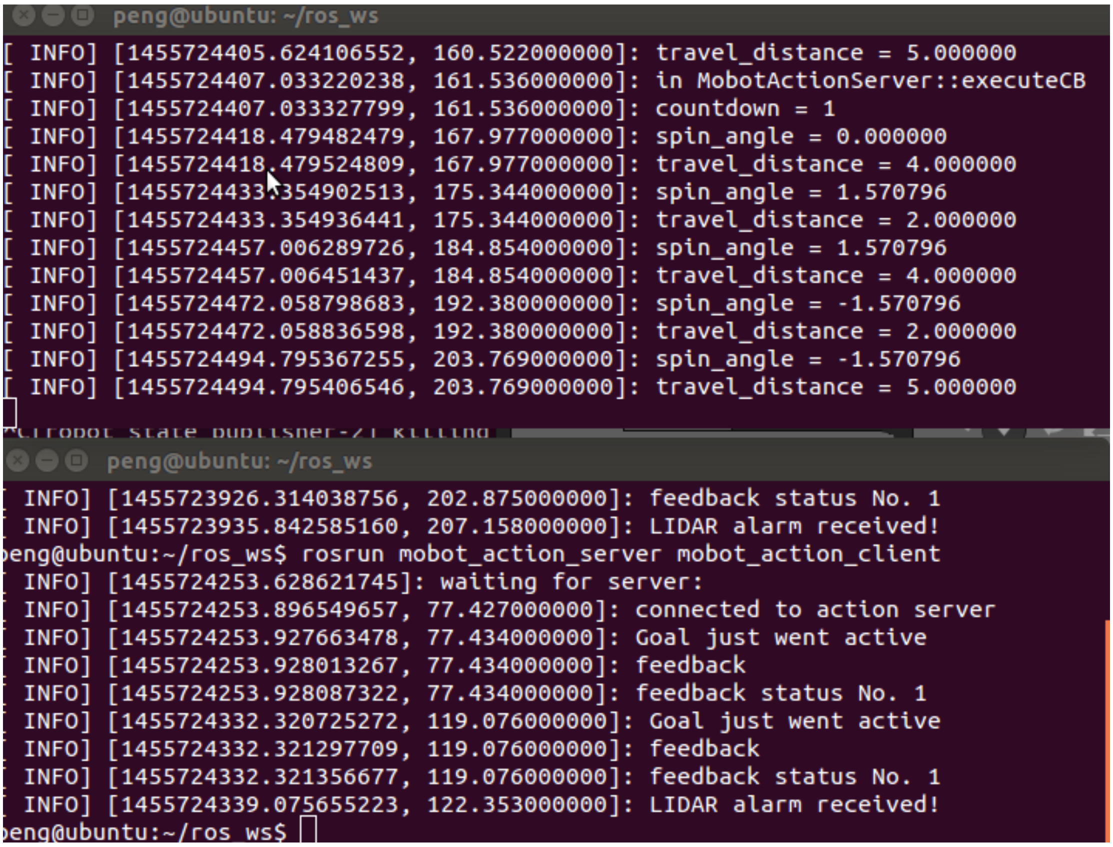
```
   while (countdown_val_>0) {
     ROS_INFO("countdown = %d",countdown_val_);

                 feedback_.fdbk = countdown_val_; // populate feedback message with current countdown
value
                         as_.publishFeedback(feedback_);
                 // excute the movement
                 for(int i = 0; i < num_angle; i++) {
                   do_spin(spin_angle[i]); // carry out this incremental action
                   do_move(travel_distance[i]); // carry out this incremental action
                   ROS_INFO("spin_angle = %f", spin_angle[i]);
                   ROS_INFO("travel_distance = %f", travel_distance[i]);
                 }

                 do_halt();

                         ros::Subscriber alarm_subscriber = nh_.subscribe("lidar_alarm",1,alarmCallback);
     // each iteration, check if cancellation has been ordered
     if (g_lidar_alarm){
       ROS_WARN("goal cancelled!");
       do_halt();
       result_.output = countdown_val_;
       as_.setAborted(result_); // tell the client we have given up on this goal; send the result message
as well
       return; // done with callback
                         }

                 //if here, then goal is still valid; provide some feedback
                 // feedback_.fdbk = countdown_val_; // populate feedback message with current
countdown value
                 // as_.publishFeedback(feedback_); // send feedback to the action client that requested
this goal
     countdown_val_--; //decrement the timer countdown
     timer.sleep(); //wait 1 sec between loop iterations of this timer
   }
```

The server receives the path information and then stores it into spin_angle and travel_distance to drive the robot moving.

Meanwhile the server sends back a feedback to the client.

4) lidar alarm

The lidar alarm here plays a role to inform the robot when to stop or cancel the mission. The following screen shot is an example when a lidar alarm is received and the client stops working.



```
[ INFO] [1455724405.624106552, 160.522000000]: travel_distance = 5.000000
[ INFO] [1455724407.033220238, 161.536000000]: in MobotActionServer::executeCB
[ INFO] [1455724407.033327799, 161.536000000]: countdown = 1
[ INFO] [1455724418.479482479, 167.977000000]: spin_angle = 0.000000
[ INFO] [1455724418.479524809, 167.977000000]: travel_distance = 4.000000
[ INFO] [1455724433.354902513, 175.344000000]: spin_angle = 1.570796
[ INFO] [1455724433.354936441, 175.344000000]: travel_distance = 2.000000
[ INFO] [1455724457.006289726, 184.854000000]: spin_angle = 1.570796
[ INFO] [1455724457.006451437, 184.854000000]: travel_distance = 4.000000
[ INFO] [1455724472.058798683, 192.380000000]: spin_angle = -1.570796
[ INFO] [1455724472.058836598, 192.380000000]: travel_distance = 2.000000
[ INFO] [1455724494.795367255, 203.769000000]: spin_angle = -1.570796
[ INFO] [1455724494.795406546, 203.769000000]: travel_distance = 5.000000
```

```
 INFO] [1455723926.314038756, 202.875000000]: feedback status No. 1
 INFO] [1455723935.842585160, 207.158000000]: LIDAR alarm received!
peng@ubuntu:~/ros_ws$ rosrun mobot_action_server mobot_action_client
 INFO] [1455724253.628621745]: waiting for server:
 INFO] [1455724253.896549657, 77.427000000]: connected to action server
 INFO] [1455724253.927663478, 77.434000000]: Goal just went active
 INFO] [1455724253.928013267, 77.434000000]: feedback
 INFO] [1455724253.928087322, 77.434000000]: feedback status No. 1
 INFO] [1455724332.320725272, 119.076000000]: Goal just went active
 INFO] [1455724332.321297709, 119.076000000]: feedback
 INFO] [1455724332.321356677, 119.076000000]: feedback status No. 1
 INFO] [1455724339.075655223, 122.353000000]: LIDAR alarm received!
peng@ubuntu:~/ros_ws$
```

2. Example use

roslaunch gazebo_ros empty_world.launch

roslaunch mobot_urdf mobot_w_lidar.launch

rosrun mobot_lidar_alarm mobot_lidar_alarm

rosrun mobot_action_server mobot_action_server_w_fdbk

rosrun mobot_action_server mobot_action_client

Feb. 17, 2016