EECS 476 Mobile Robotics

PS 4

Peng Xu

[pxx37@case.edu](mailto:pxx37@case.edu)

1. Main Methods

in client: alarmCallback()

void alarmCallback(const std\_msgs::Bool& alarm\_msg) {

bool g\_lidar\_alarm = alarm\_msg.data; //make the alarm status global, so main() can use it

mobot\_action\_server::pathGoal goal;

ROS\_INFO("LIDAR alarm = %d", g\_lidar\_alarm);

if (g\_lidar\_alarm) {

ROS\_INFO("LIDAR alarm received!");

action\_client->cancelGoal();

action\_client->waitForResult(); // wait forever...

ROS\_INFO("turning around!");

//commandSquarePath();

std::vector<geometry\_msgs::PoseStamped> poses(1);

poses.at(0).pose.position.x = 0;

poses.at(0).pose.position.y = 0;

poses.at(0).pose.orientation = convertPlanarPhi2Quaternion(PI\*3/4);

goal.path.poses = poses;

action\_client->sendGoal(goal);

action\_client->waitForResult(); // wait forever...

// new goal

poses.resize(4);

poses.at(0).pose.position.x = 0;

poses.at(0).pose.position.y = 5;

poses.at(0).pose.orientation = convertPlanarPhi2Quaternion(0);

poses.at(1).pose.position.x = 0;

poses.at(1).pose.position.y = 5;

poses.at(2).pose.position.x = 0;

poses.at(2).pose.position.y = 5;

poses.at(3).pose.position.x = 0;

poses.at(3).pose.position.y = 5;

goal.path.poses = poses;

action\_client->sendGoal(goal, &doneCb, &activeCb, &feedbackCb);

}

}

in server:

void MobotActionServer::executeCB(const actionlib::SimpleActionServer<mobot\_action\_server::pathAction>::GoalConstPtr& goal) {

ROS\_INFO("in executeCB");

nav\_msgs::Path path = goal->path;

ros::Rate timer(1.0); // 1Hz timer

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double spin\_angle;

double travel\_distance;

geometry\_msgs::Pose pose\_desired;

int npts = path.poses.size();

ROS\_INFO("received path request with %d poses",npts);

for (int i = 0; i < npts; i++) { //visit each subgoal

// each iteration, check if cancellation has been ordered

if (as\_.isPreemptRequested()){

ROS\_WARN("goal cancelled!");

result\_.is\_alldone.data = false;

do\_halt();

as\_.setAborted(result\_); // tell the client we have given up on this goal; send the result message as well

return; // done with callback

}

// odd notation: drill down, access vector element, drill some more to get pose

pose\_desired = path.poses[i].pose; //get first pose from vector of poses

feedback\_.des\_pose = pose\_desired;

feedback\_.is\_finished.data = false; // populate feedback message with current countdown value

as\_.publishFeedback(feedback\_); // send feedback to the action client that requested this goal

get\_yaw\_and\_dist(current\_pose\_, pose\_desired, travel\_distance, spin\_angle);

do\_halt();

spin\_angle = min\_spin(spin\_angle);// but what if this angle is > pi? then go the other way

do\_spin(spin\_angle); // carry out this incremental action

current\_pose\_.orientation = pose\_desired.orientation; // assumes got to desired orientation precisely

do\_move(travel\_distance); // carry out this incremental action

current\_pose\_.position = pose\_desired.position; // assumes got to desired orientation precisely

feedback\_.is\_finished.data = true; // populate feedback message with current countdown value

as\_.publishFeedback(feedback\_); // send feedback to the action client that requested this goal

ros::spinOnce();

timer.sleep();

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

result\_.is\_alldone.data = true; //value should be zero, if completed countdown

as\_.setSucceeded(result\_); // return the "result" message to client, along with "success" status

}

2. Experiment Design

Basically I intend to make the robot do a square move if it doesn't encounter a block.

While it gets an alarm that it faces a block within 1.2 meters, it will cancel the current goal, turn 270 degrees and restart the square move, which are shown in alarmCallback().

3. The youtube video

<https://youtu.be/p-idbc-d6C4>