```
/*************************************************/
/*This code was used at WRS.                     */
/*You can clone it from DemuLab's private repository*/
/*"https://github.com/demulab/happy_burger3.git"    */
/*************************************************/
#include <ros/ros.h>
#include <tf/tf.h>
#include <geometry_msgs/Twist.h>
#include <geometry_msgs/Pose2D.h>
#include <nav_msgs/Odometry.h>
#include <math.h>
#include <std_msgs/Bool.h>
#include <sstream>
#include <iostream>
#include "arm.hpp"
#include <vector>

using namespace std;

geometry_msgs::Pose2D current_pose;
ros::Publisher pose2d_pub;
ros::Publisher twist_pub;
geometry_msgs::Twist twist;

struct MyPose {
  double x;
  double y;
  double yaw;
};

MyPose waypoint[] = {
  { 0.40,  0.00,  0.0 * M_PI/180},//0
  { 1.05,  0.00,  -90 * M_PI/180},//1
  { 1.05, -0.80,  -90 * M_PI/180},//2
  { 1.05, -0.80, -180 * M_PI/180},//3
  { 0.40, -0.80, -180 * M_PI/180},//4
  { 0.00, -0.80,  0.0 * M_PI/180},//5
  { 0.40, -0.80,  0.0 * M_PI/180},//6
  { 1.05, -0.80,   90 * M_PI/180},//7
  { 1.05,   0.0,  180 * M_PI/180},//8
  { 0.90,   0.0,  180 * M_PI/180},//9
  { -0.1,   0.1,  0.0 * M_PI/180},//10
  {  999,   999,            999}};

// cleaning right   { 0.36809, 0.3619,-1.31161, 1.77482,-0.57369}
// super man        {-0.04136,-0.9142,-0.04451,-0.66724,-0.57369}
// cleaning left    { 2.90684, 0.4632,-1.09066, 1.57334,-0.57369}
std::vector<std::vector<double>>  armpose ={
  { 0.36809, 0.3619,-1.31161, 1.77482,-0.57369},//0
```

```
  { 0.36809, 0.3619,-1.31161, 1.77482,-0.57369},//1
  { 0.36809, 0.3619,-1.31161, 1.77482,-0.57369},//2
  { 0.36809, 0.3619,-1.31161, 1.77482,-0.57369},//3
  {-0.04136,-0.9142,-0.04451,-0.66724,-0.57369},//4
  {-0.04136,-0.9142,-0.04451,-0.66724,-0.57369},//5
  { 2.90684, 0.4632,-1.09066, 1.57334,-0.57369},//6
  { 2.90684, 0.4632,-1.09066, 1.57334,-0.57369},//7
  { 2.90684, 0.4632,-1.09066, 1.57334,-0.57369},//8
  {-0.04136,-0.9142,-0.04451,-0.66724,-0.57369},//9
  {-0.04136,-0.9142,-0.04451,-0.66724,-0.57369},//10
  {-0.04136,-0.9142,-0.04451,-0.66724,-0.57369},//11
};

const double rot_vel = 0.3;    // angular velocity [rad/s]
const double linear_vel = 0.2; // linear  velocity [m/s]
const double kp_linear  = 0.75;  // proportional gain for lenear velocity
const double kp_angular = 1.0;  // proportional gain for lenear velocity
const double ki_angular = 0.05;  // proportional gain for lenear velocity
const double rot_vel_i = 0.1;

void odomCallback(const nav_msgs::OdometryConstPtr& msg)
{
  // linear position
  current_pose.x = msg->pose.pose.position.x;
  current_pose.y = msg->pose.pose.position.y;

  // quaternion to RPY conversion
  tf::Quaternion q(
    msg->pose.pose.orientation.x,
    msg->pose.pose.orientation.y,
    msg->pose.pose.orientation.z,
    msg->pose.pose.orientation.w);
  tf::Matrix3x3 m(q);
  double roll, pitch, yaw;
  m.getRPY(roll, pitch, yaw);

  // angular position
  current_pose.theta = yaw;
  pose2d_pub.publish(current_pose);
}

// Set the direction
void gotoDirection(int no)
{
  double thresh = 1.0 * M_PI/180.0;
  double diff = thresh + 1; // make diff is bigger than thresh

  ros::Rate loop_rate(100);
  double angle_tmp_i = 0;
  twist.linear.x = 0; // linear velocity is 0, i.e., only rotate
```

```
100   while (fabs(diff) > thresh) {
101     diff = waypoint[no].yaw - current_pose.theta;
102     if (diff >   M_PI) diff -= 2 * M_PI;
103     else if (diff < - M_PI) diff += 2 * M_PI;
104     // cout << "dir diff:" << diff << endl;
105     // p
106     double angle_tmp_p = kp_angular * diff;
107     // i
108     angle_tmp_i += ki_angular*diff;
109     if (angle_tmp_i >= rot_vel_i ) angle_tmp_i=rot_vel_i;
110     else if (angle_tmp_i <= -rot_vel_i) angle_tmp_i=-rot_vel_i;
111
112     double angle_tmp = angle_tmp_p + angle_tmp_i;
113     if (angle_tmp >= rot_vel ) angle_tmp=rot_vel;
114     else if (angle_tmp <= -rot_vel) angle_tmp=-rot_vel;
115
116     twist.angular.z =  angle_tmp;
117
118     twist_pub.publish(twist);
119     ros::spinOnce();
120     loop_rate.sleep();
121   }
122   twist.linear.x  = linear_vel;
123   twist.angular.z = 0;
124   twist_pub.publish(twist);
125   ros::spinOnce();
126 }
127
128 // Go to the next waypoint
129 void gotoPosition(int no)
130 {
131   double thresh = 0.01; // [m]
132   // initial diff_dist should be bigger than thresh
133   double diff_dist = thresh + 1;
134   twist.linear.x = linear_vel;
135
136   ros::Rate loop_rate(100);
137
138   double angle_tmp_i = 0;
139   while (fabs(diff_dist) > thresh) {
140     double diff_x = waypoint[no].x - current_pose.x;
141     double diff_y = waypoint[no].y - current_pose.y;
142     double diff_theta = atan2(diff_y, diff_x) - current_pose.theta;
143     diff_dist = sqrt(diff_x * diff_x + diff_y * diff_y);
144     cout << "gotoPosition" << diff_dist << endl;
145     cout << "diff dist:" << diff_dist << "[m]" << endl;
146     cout << "current_pose.x:" << current_pose.x << "[m]" << endl;
147     cout << "current_pose.y:" << current_pose.y << "[m]" << endl;
148     cout << "current_pose.theta:" << current_pose.theta << "[rad]" << endl;
149     if (diff_theta >   M_PI) diff_theta -= 2 * M_PI;
```

```
150     else if (diff_theta < - M_PI) diff_theta += 2 * M_PI;
151     //maximum velosity
152     double angle_tmp_p = kp_angular * diff_theta;
153     // i
154     angle_tmp_i += ki_angular*diff_theta;
155     if (angle_tmp_i >= rot_vel_i ) angle_tmp_i=rot_vel_i;
156     else if (angle_tmp_i <= -rot_vel_i) angle_tmp_i=-rot_vel_i;
157
158     double angle_tmp = angle_tmp_p + angle_tmp_i;
159     if (angle_tmp >= rot_vel ) angle_tmp=rot_vel;
160     else if (angle_tmp <= -rot_vel) angle_tmp=-rot_vel;
161
162     twist.angular.z =  angle_tmp;
163
164    //maximum velosity
165     double dist_tmp = kp_linear * diff_dist;
166     if (dist_tmp >= linear_vel ) dist_tmp=linear_vel;
167     else if (dist_tmp <= -linear_vel) dist_tmp=-linear_vel;
168     twist.linear.x  = dist_tmp;
169
170     twist_pub.publish(twist);
171     ros::spinOnce();
172     loop_rate.sleep();
173   }
174   twist.angular.z = 0;
175   twist.linear.x  = 0;
176   twist_pub.publish(twist);
177   ros::spinOnce();
178   ROS_INFO("Arrived at WP %d",no);
179   sleep(1); // [s]
180 }
181
182 void gotoWaypoint(int no)
183 {
184   gotoPosition(no);
185   gotoDirection(no);
186 }
187
188
189
190 int main(int argc, char **argv)
191 {
192   ROS_INFO("Start");
193
194   //magic, don't think about here.
195   ros::init(argc, argv, "happy_navi");
196   ros::NodeHandle nh;
197   ros::Subscriber odom_sub = nh.subscribe("odom", 1, odomCallback);
198   twist_pub = nh.advertise<geometry_msgs::Twist>("cmd_vel",1000);
199   pose2d_pub = nh.advertise<geometry_msgs::Pose2D>("roomba_pose2d", 1);
```

```
  ros::Rate rate(10);

  Arm arm(&nh);

  //initializing way point.
z axis.
  int next_wp = 0; // Next waypoint
  int next_ap = 0; // Next armpoint
  sleep(1);
  //initializing arm position
  arm.armPos(armpose[11]);
  while(ros::ok() && arm.moveCheck()){
    ros::spinOnce();
    arm.cycle();
  }

  //main while
  while (ros::ok() && waypoint[next_wp].x != 999) {
    ros::spinOnce();
    //way point running
    ROS_INFO("Go to WP %d",next_wp);
    gotoWaypoint(next_wp);
    arm.armPos(armpose[next_ap]);

    while(ros::ok() && arm.moveCheck()){
      ros::spinOnce();
      arm.cycle();
    }
    next_wp++;
    next_ap++;
  }

  twist.angular.z = 0;
  twist.linear.x  = 0;
  twist_pub.publish(twist);
  ROS_INFO("Mission complete!");
  sleep(3); // [s]
  return 0;
}
```