**−Table of Contents**

* [Introduction](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:final_project#introduction)
* [Gmapping](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:final_project#gmapping)
* [launch files](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:final_project#launch_files)
* [my\_subsumption.cpp (Main)](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:final_project#my_subsumptioncpp_main)
* [Go to red beacon](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:final_project#go_to_red_beacon)
* [Order](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:final_project#order)
* [Deliver order](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:final_project#deliver_order)
* [Pick up order](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:final_project#pick_up_order)

**Introduction**

For this project I used 4 noded subsumption algorithm. I implemented each node in a different package and combined them in another master backage my\_subsumption.

For this project I did

* Basic Scenario
* Localization
* Grasping Ability
* Multi Agent Capability
* Power Consumption

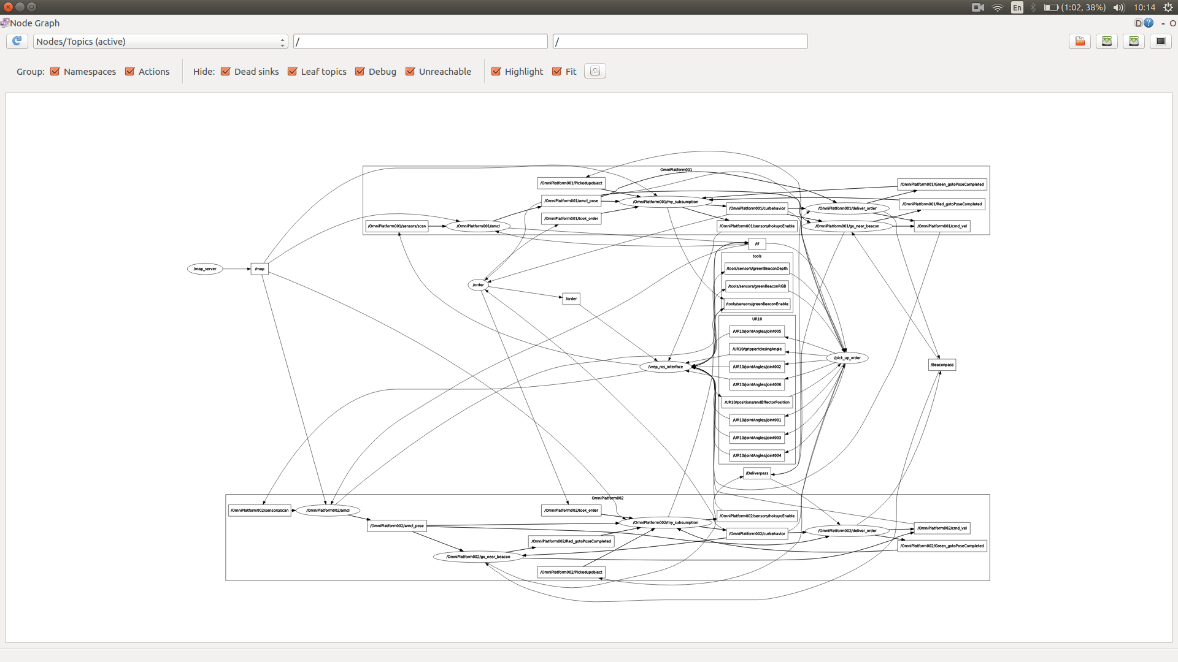
Since I didn't implement any exploration I only used scan data for amcl point-cloud and green beacon for grasping.The mentioned 4 packages are :

* go\_near\_beacon : Go to red

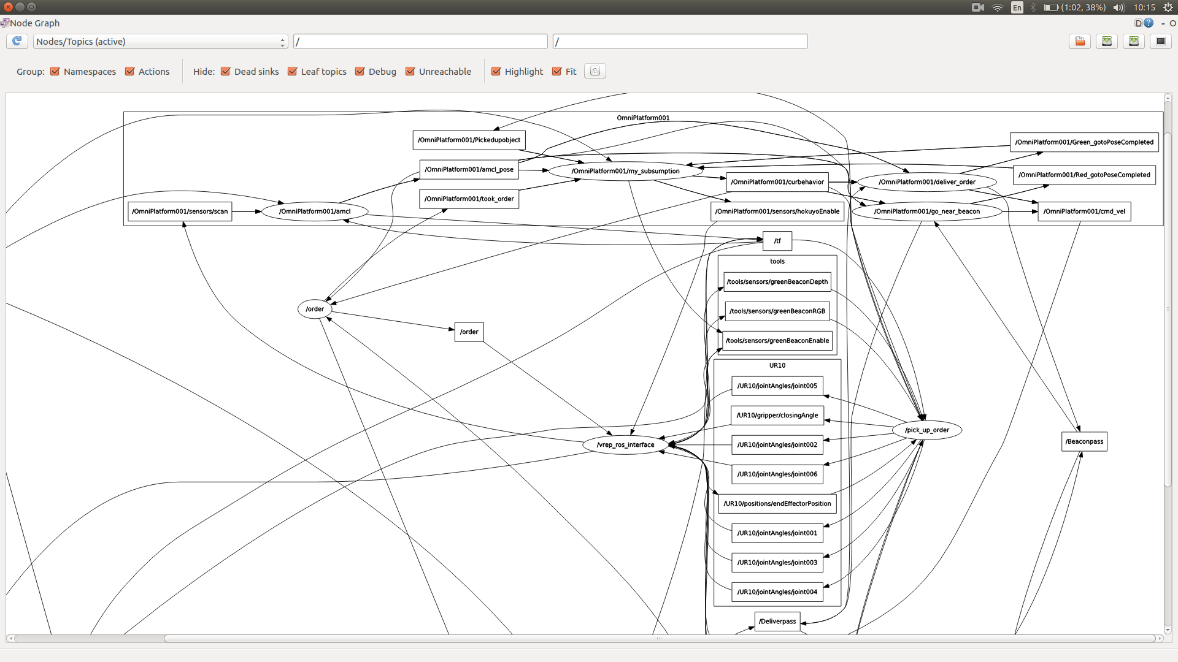
beacon

* order : Request Object from redbeacon
* deliver\_order :Carry object to green beacon
* pick\_up\_order : Pick and place object with UR10

I implemented each algorithm with its own packages. And they communicate each other via published/subscribed topics .



Robot1 closeup:



AS you can see /Omniplatform001/my\_subsumption controls /Omniplatform001/curbehavior node which is the selection topic for subsumption.

Full code can be found at <https://drive.google.com/open?id=1vTguIQjXRus00YJx8jV_I1QCC-ILY6Ps>.

Formun Üstü

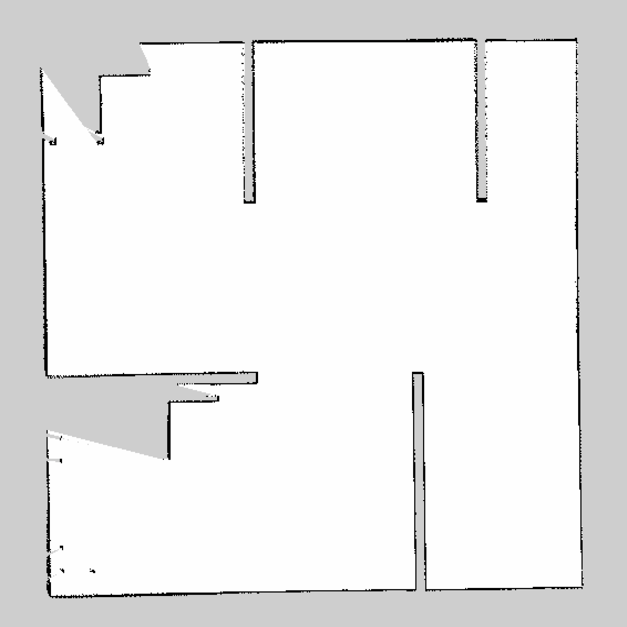
Formun Altı

**Gmapping**

I modified the gmapping algorithm in nav stack package in assignment 7. Then I controlled the robot with keyboard and save the map.

https://youtu.be/Y\_oXK3SXJIs

**MAP**



Formun Üstü

Formun Altı

**launch files**

final\_project.launch :

<launch>

<master auto="start"/>

<param name="/use\_sim\_time" value="true" />

<arg name="map\_file" default="/home/alpsark/ros-ws/src/nav\_stack\_tuning/launch/map.yaml"/>

<!-- Map server -->

<node name="map\_server" pkg="map\_server" type="map\_server" args="$(arg map\_file)" >

<param name="frame\_id" value ="/map"/>

</node>

<remap from="map" to="/map"/>

<!--node name="rviz" pkg="rviz" type="rviz" output="screen" /-->

<!--My packages -->

<group ns="/OmniPlatform001/">

<param name="tf\_prefix" value="robot1\_tf" />

<remap from="map" to="/map"/>

<param name = "amcl/initial\_pose\_x" value="0.0" />

<param name = "amcl/initial\_pose\_y" value="2.0" />

<param name = "amcl/base\_frame\_id" value="OmniPlatform001\_base" />

<include file = "/home/alpsark/ros-ws/src/my\_subsumption/launch/robot.launch" >

<arg name = "robot\_name" value="OmniPlatform001"/>

<arg name = "scan\_name" value="/OmniPlatform001/sensors/scan"/>

</include>

<node name="go\_near\_beacon" pkg="go\_near\_beacon" type="go\_near\_beacon" output="screen">

<param name="prefix" value ="/OmniPlatform001/"/>

</node>

<node name="deliver\_order" pkg="deliver\_order" type="deliver\_order" output="screen">

<param name="prefix" value ="/OmniPlatform001/"/>

</node>

<node name="my\_subsumption" pkg="my\_subsumption" type="my\_subsumption" output="screen" >

<param name="prefix" value ="/OmniPlatform001/"/>

</node>

</group>

<group ns="/OmniPlatform002/">

<param name="tf\_prefix" value="robot2\_tf" />

<remap from="map" to="/map"/>

<include file = "/home/alpsark/ros-ws/src/my\_subsumption/launch/robot.launch" >

<arg name = "robot\_name" value="OmniPlatform002"/>

<arg name = "scan\_name" value="/OmniPlatform002/sensors/scan"/>

</include>

<param name = "amcl/initial\_pose\_x" value="1.0" />

<param name = "amcl/initial\_pose\_y" value="2.0" />

<param name = "amcl/base\_frame\_id" value="OmniPlatform002\_base" />

<node name="go\_near\_beacon" pkg="go\_near\_beacon" type="go\_near\_beacon" output="screen">

<param name="prefix" value ="/OmniPlatform002/"/>

</node>

<node name="deliver\_order" pkg="deliver\_order" type="deliver\_order" output="screen">

<param name="prefix" value ="/OmniPlatform002/"/>

</node>

<node name="my\_subsumption" pkg="my\_subsumption" type="my\_subsumption" output="screen" >

<param name="prefix" value ="/OmniPlatform002/"/>

</node>

</group>

<node name="order" pkg="order" type="order" output="screen"/>

<node name="pick\_up\_order" pkg="pick\_up\_order" type="pick\_up\_order" output="screen" />

</launch>

robot.launch :

<launch>

<arg name="robot\_name"/>

<arg name="scan\_name" />

<master auto="start"/>

<param name="/use\_sim\_time" value="true" />

<!-- Localization -->

<node name="amcl" pkg="amcl" type="amcl" output="screen">

<param name="use\_map\_topic" value="true"/>

<remap from = "static\_map" to="/static\_map"/>

<param name = "global\_frame\_id" value="/map" />

<remap from="scan" to="$(arg scan\_name)"/>

<param name="odom\_frame\_id" value="odom"/>

<remap from="map" to="/map"/>

<!--Diff for 2 wheel omni for omnidirectional robots either "diff", "omni", "diff-corrected" or "omni-corrected"-->

<param name="odom\_model\_type" value="omni"/>

<!-- Specifies the expected noise in odometry's rotation estimate from the rotational component of the robot's motion. Default="0.2"-->

<param name="odom\_alpha1" value="0.2"/>

<!-- Specifies the expected noise in odometry's rotation estimate from translational component of the robot's motion. Default="0.2"-->

<param name="odom\_alpha2" value="0.2"/>

<!-- Specifies the expected noise in odometry's translation estimate from the translational component of the robot's motion. Default = "0.2" -->

<param name="odom\_alpha3" value="0.2"/>

<!-- Specifies the expected noise in odometry's translation estimate from the rotational component of the robot's motion. Default="0.2"-->

<param name="odom\_alpha4" value="0.2"/>

<!--Transaltiion related noise paramater (only used if model is "omni") Default="0.2"-->

<param name="odom\_alpha5" value="0.1"/>

<!-- Maximum rate (Hz) at which scans and paths are published for visualization, -1.0 to disable. -->

<param name="gui\_publish\_rate" value="10.0"/>

<!-- How many evenly-spaced beams in each scan to be used when updating the filter. Default="30"-->

<param name="laser\_max\_beams" value="80"/>

<param name="laser\_max\_range" value="5.0"/>

<param name="min\_particles" value="500"/>

<param name="max\_particles" value="3000"/>

<!-- Maximum error between the true distribution and the estimated distribution. Default="0.01"-->

<param name="kld\_err" value="0.05"/>

<!--Upper standard normal quantile for (1 - p), where p is the probability that the error on the estimated distrubition will be less than kld\_err. Default="0.99"-->

<param name="kld\_z" value="0.99"/>

<!-- Mixture weight for the z\_hit part of the model. Default="0.95"-->

<param name="laser\_z\_hit" value="0.5"/>

<!-- Mixture weight for the z\_short part of the model. Default="0.1"-->

<param name="laser\_z\_short" value="0.05"/>

<!-- Mixture weight for the z\_max part of the model. Default="0.05"-->

<param name="laser\_z\_max" value="0.05"/>

<!-- Mixture weight for the z\_rand part of the model. Default="0.05"-->

<param name="laser\_z\_rand" value="0.5"/>

<!-- Standard deviation for Gaussian model used in z\_hit part of the model. Default="0.2"-->

<param name="laser\_sigma\_hit" value="0.2"/>

<!-- Exponential decay parameter for z\_short part of model. Default="0.1"-->

<param name="laser\_lambda\_short" value="0.1"/>

<!--Which model to use, either beam, likelihood\_field, or likelihood\_field\_prob (same as likelihood\_field but incorporates the beamskip feature, if enabled). -->

<param name="laser\_model\_type" value="likelihood\_field"/>

<!-- < Maximum distance to do obstacle inflation on map, for use in likelihood\_field model. default= 2.0 meters"/> -->

<param name="laser\_likelihood\_max\_dist" value="2.0"/>

<!-- Translational movement required before performing a filter update. Default="0.2"-->

<param name="update\_min\_d" value="0.25"/>

<!-- Rotational movement required before performing a filter update. Default="pi/6.0 rad"-->

<param name="update\_min\_a" value="0.2"/>

<!-- Number of filter updates required before resampling. Default="2"-->

<param name="resample\_interval" value="1"/>

<!-- Time with which to post-date the transform that is published, to indicate that this transform is valid into the future. Default:0.1-->

<param name="transform\_tolerance" value="0.2"/>

<!--Exponential decay rate for the slow average weight filter, used in deciding when to recover by adding random poses. A good value might be 0.001.-->

<param name="recovery\_alpha\_slow" value="0.0"/>

<!--Exponential decay rate for the fast average weight filter, used in deciding when to recover by adding random poses. A good value might be 0.1. -->

<param name="recovery\_alpha\_fast" value="0.0"/>

<param name="initial\_pose\_a" value="-1.57079"/>

</node>

</launch>

Formun Üstü

Formun Altı

**my\_subsumption.cpp (Main)**

subscriptions :

* /amcl\_pose for localization
* Beaconpass with Go to red beacon node of other robot
* map map data (not really used , it was added to implement exploration algorithm)
* took\_order communication with the order package
* Red\_gotoPoseCompleted communication with the go red beacon package
* Green\_gotoPoseCompleted communication with the deliver order package
* Pickedupobject communication with the pick up order package

publications :

* /curbehavior every other node
* prefix + “sensors/hokuyoEnable” open/close hokuyo scan sensors
* /tools/sensors/greenBeaconEnable open/close green beacon image and depth sensors

#include <ros/ros.h>

#include <std\_msgs/Float64.h>

#include <std\_msgs/Bool.h>

#include <std\_msgs/Float32.h>

#include <std\_msgs/String.h>

#include <tf/transform\_listener.h>

#include <tf/tf.h>

#include <sensor\_msgs/LaserScan.h>

#include <nav\_msgs/OccupancyGrid.h>

#include <geometry\_msgs/Twist.h>

#include <geometry\_msgs/PoseWithCovarianceStamped.h>

#include <geometry\_msgs/PoseWithCovariance.h>

#include <geometry\_msgs/Point.h>

#include <geometry\_msgs/Pose.h>

#include <geometry\_msgs/Twist.h>

#include <visualization\_msgs/MarkerArray.h>

#include <visualization\_msgs/Marker.h>

#include <nav\_msgs/OccupancyGrid.h>

#include <nav\_msgs/GetMap.h>

#include <vector>

#include "map\_helper.h"

#include "image\_helper.h"

#define PI 3.1415926535897932384626433f

/\*\*

\* The node which runs simple subsumption architecture

\* @param argc

\* @param argv

\* @return

\*/

//0.5 safe

std::string prev\_behavior;

ros::Publisher scanEnable ;

ros::Publisher beaconEnable ;

ros::Publisher CurrentBehavior ;

MapHelper mapHelper;

ImageHelper imageHelper;

float robotX;

float robotY;

float robotYaw;

float redx;

float redy;

float greenx;

float greeny;

bool took\_order;

bool isredgotoposecompleted;

bool isgreengotoposecompleted;

bool ispickedupobject;

std::string prefix;

bool go\_near\_beacon(){

// false ;

if(!isredgotoposecompleted){

return true ;

}else{return false;}

}

bool order(){

//return false ;

if(!took\_order){

return true ;

}else{return false;}

}

bool deliver\_order(){

//return false ;

if(!isgreengotoposecompleted){

return true ;

}else{return false;}

}

bool pick\_up\_order(){

if(!ispickedupobject){

return true ;

}else{

took\_order = false;

isgreengotoposecompleted = false;

isredgotoposecompleted = false;

ispickedupobject = false;

return false;}

}

std\_msgs::String Choosebehavior(){//choose next beahvior

std\_msgs::Bool data;

data.data = true;

//beaconEnable.publish(data);

std\_msgs::String behaviour;

if(go\_near\_beacon()){

if(prev\_behavior != "go\_near\_beacon"){

std::cout <<prefix + "ChoosenBehavior is go\_near\_red\_beacon" <<std::endl;

}

scanEnable.publish(data);

data.data = false;

beaconEnable.publish(data);

behaviour.data = "go\_near\_beacon" ;

prev\_behavior = "go\_near\_beacon";

return behaviour ;

}

if(order()){

if(prev\_behavior != "order"){

std::cout <<prefix + "ChoosenBehavior is order" <<std::endl;

}

data.data = false;

scanEnable.publish(data);

beaconEnable.publish(data);

behaviour.data = "order" ;

prev\_behavior = "order";

return behaviour ;

}

if(deliver\_order()){

scanEnable.publish(data);

data.data = false;

beaconEnable.publish(data);

if(prev\_behavior != "deliver\_order"){

std::cout << prefix + "ChoosenBehavior is deliver\_order" <<std::endl;

}

behaviour.data = "deliver\_order" ;

prev\_behavior = "deliver\_order";

return behaviour ;

}

if(pick\_up\_order()){

if(prev\_behavior != "pick\_up\_order"){

data.data = false;

scanEnable.publish(data);

data.data = true;

beaconEnable.publish(data);

std::cout << prefix + "ChoosenBehavior is pick\_up\_order" <<std::endl;

}

behaviour.data = "pick\_up\_order" ;

prev\_behavior = "pick\_up\_order";

return behaviour ;

}

}

void updatePose(const geometry\_msgs::PoseWithCovarianceStamped& NewPose){

geometry\_msgs::PoseWithCovariance RobotPosewithcovariance = NewPose.pose;

geometry\_msgs::Pose RobotPose = RobotPosewithcovariance.pose;

robotX=RobotPose.position.x;

robotY=RobotPose.position.y;

robotYaw=tf::getYaw(RobotPose.orientation);

//std::cout <<prefix << "PoseUpdated: " << robotX << "," << robotY << "," << robotYaw <<std::endl;

}

//collects sensor data and decides behavior

//avoid\_object

//explore

//go\_near\_Beacon

//get\_order

//deliver\_order

//pick\_up\_order

void findBeaconLocation(float th,std::string color){

float iteration = 0.1 ;

float i = 0;

while(true){

if( mapHelper.isReallyOccupied(robotX+ i\*cos(robotYaw-th) , robotY+ i\*sin(robotYaw-th) )) {

if(color == "green"){

greenx = robotX+ (i)\*cos(robotYaw-th);

greeny = robotY+ (i)\*sin(robotYaw-th);

}else if(color == "red"){

redx = robotX + (i)\*cos(robotYaw-th);

redy = robotY+ (i)\*sin(robotYaw-th);

}

return;

}

i += iteration ;

}

}

void took\_func (const std\_msgs::Bool::ConstPtr& BoolPointer ){

if(BoolPointer->data ){

took\_order = true;

}else{

took\_order = false;

}

}

void Red\_gotoPoseCompleted(const std\_msgs::Bool::ConstPtr& BoolPointer ){

if(BoolPointer->data ){

isredgotoposecompleted = true;

}else{

isredgotoposecompleted = false;

}

}

void Green\_gotoPoseCompleted(const std\_msgs::Bool::ConstPtr& BoolPointer ){

if(BoolPointer->data ){

isgreengotoposecompleted = true;

}else{

isgreengotoposecompleted = false;

}

}

void pickedupobject\_func(const std\_msgs::Bool::ConstPtr& BoolPointer ){

if(BoolPointer->data ){

ispickedupobject = true;

}else{

ispickedupobject = false;

}

}

int main(int argc, char\*\* argv)

{

prev\_behavior = "no behavior";

ros::init(argc,argv,"my\_subsumption" );

ros::NodeHandle nh;

ros::Rate rate(30);

// read parameters

ros::NodeHandle private\_nh("~");

if (!private\_nh.getParam("prefix", prefix)){

//initialx = 0;

}

// construct publishers and subscribers

scanEnable = nh.advertise<std\_msgs::Bool>(prefix + "sensors/hokuyoEnable", 1);

CurrentBehavior = nh.advertise<std\_msgs::String>(prefix + "curbehavior", 1);

//ros::Publisher took\_orderPub = nh.advertise<std\_msgs::Bool>("/took\_order", 1);

beaconEnable = nh.advertise<std\_msgs::Bool>("/tools/sensors/greenBeaconEnable", 1);

//ros::Subscriber laserSubs = nh.subscribe("/OmniPlatform001/sensors/scan", 10, processLaserScan);

ros::Subscriber mapSubs = nh.subscribe("/map", 10, &MapHelper::processMap, &mapHelper);

ros::Subscriber poseSubs = nh.subscribe(prefix + "amcl\_pose", 10, updatePose); //geometry\_msgs/PoseWithCovarianceStamped

//ros::Subscriber poseSubs = nh.subscribe("/amcl\_pose", 10, updatePose); //geometry\_msgs

ros::Subscriber took\_orderSubs =nh.subscribe(prefix + "took\_order",1,took\_func);

ros::Subscriber RedcompleteSubs =nh.subscribe(prefix + "Red\_gotoPoseCompleted",1,Red\_gotoPoseCompleted);

ros::Subscriber GreencompleteSubs =nh.subscribe(prefix + "Green\_gotoPoseCompleted",1,Green\_gotoPoseCompleted);

ros::Subscriber PickedupSubs =nh.subscribe(prefix + "Pickedupobject",1,pickedupobject\_func);

std\_msgs::Bool data;

data.data = true;

scanEnable.publish(data);

data.data = false;

beaconEnable.publish(data);

took\_order =false;

isredgotoposecompleted = false;

isgreengotoposecompleted = false;

ispickedupobject = false;

if (prefix =="/OmniPlatform002/"){

usleep(60 \*450000);

usleep(60 \*450000);

usleep(60 \*450000);

usleep(60 \*450000);

usleep(60 \*450000);

usleep(60 \*450000);

usleep(60 \*450000);

usleep(60 \*450000);

usleep(2 \*450000);

}

while(ros::ok())

{

CurrentBehavior.publish(Choosebehavior());

ros::spinOnce();

rate.sleep();

}

return 0;

}

Formun Üstü

Formun Altı

**Go to red beacon**

Uses: scan for localization

subscriptions :

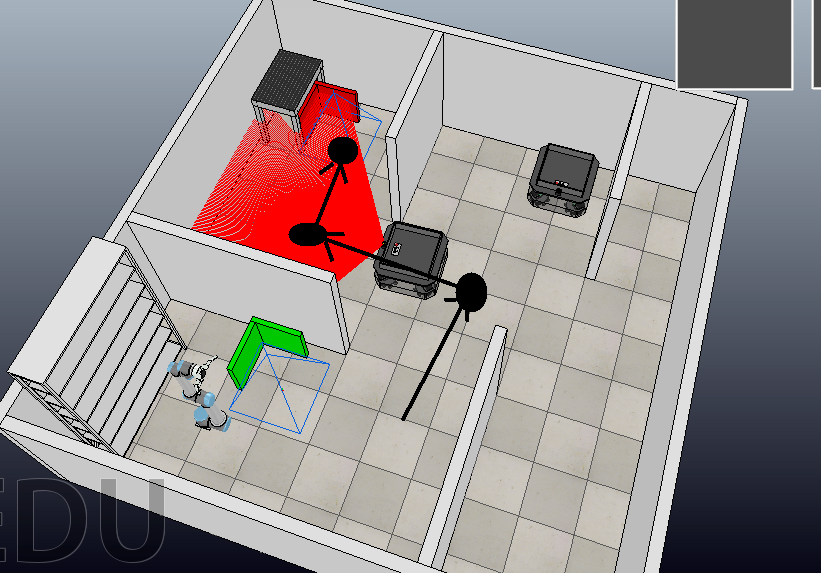
* /curbehavior with my\_subssumption node
* /amcl\_pose for localization
* Beaconpass with Go to red beacon node of other robot

publications :

* Deliverpass with deliver\_order node of other robot
* Green\_gotoPoseCompleted with my\_subssumption node
* /cmd\_vel for movement

Beaconpass and Deliverpass are the hierarchical waiting commands for multi agent system.

It goes to target via predetermined locations (black circles)



It waits for the delivery node in middle circle.

#include <ros/ros.h>

#include <std\_msgs/Bool.h>

#include <std\_msgs/String.h>

#include <sensor\_msgs/LaserScan.h>

#include <geometry\_msgs/Twist.h>

#include <geometry\_msgs/Pose.h>

#include <geometry\_msgs/Point.h>

#include <geometry\_msgs/PoseWithCovarianceStamped.h>

#include <geometry\_msgs/PoseWithCovariance.h>

#include <tf/transform\_listener.h>

#include <tf/tf.h>

#include "map\_helper.h"

#include <visualization\_msgs/MarkerArray.h>

#include <visualization\_msgs/Marker.h>

#include <nav\_msgs/OccupancyGrid.h>

#include <nav\_msgs/GetMap.h>

#include <move\_base\_msgs/MoveBaseAction.h>

#include <actionlib/client/simple\_action\_client.h>

#include <visualization\_msgs/MarkerArray.h>

#include <visualization\_msgs/Marker.h>

#include <nav\_msgs/OccupancyGrid.h>

#include <nav\_msgs/GetMap.h>

#include <std\_msgs/Bool.h>

#include <tf/transform\_listener.h>

#include <vector>

#include "MyRRTSolver.h"

#include "graph/MyVertex.h"

#include "graph/MyGraph.h"

#include "graph/MyEdge.h"

#include "image\_helper.h"

#define PI 3.1415926535897932384626433f

#define MAX\_TURN\_SPD 0.52359879

#define MIN\_TURN\_SPD 0.05

#define MAX\_LINEAR\_SPD 0.7

#define EPS 0.05

bool isbeavioractive ;

bool gotoPoseCompleted ;

bool robotposeGet ;

bool obstacle\_detected;

bool isrobot\_scan\_all\_area;

ros::Publisher completedPub;

float robotX;

float robotY;

float robotYaw;

std::string prefix;

bool behavior\_changed;

bool pass;

void updatePose(const geometry\_msgs::PoseWithCovarianceStamped& NewPose){

geometry\_msgs::PoseWithCovariance RobotPosewithcovariance = NewPose.pose;

geometry\_msgs::Pose RobotPose = RobotPosewithcovariance.pose;

robotX=RobotPose.position.x;

robotY=RobotPose.position.y;

robotYaw=tf::getYaw(RobotPose.orientation);

robotposeGet = true ;

}

float normalizeRad(float rad)

{

while(rad>PI)

{

rad=rad-2\*PI;

}

while(rad<-PI)

{

rad=rad+2\*PI;

}

return rad;

}

geometry\_msgs::Twist gotoPose(float targetPosex , float targetPosey , float targetYaw){

// create velocity vector

geometry\_msgs::Twist moveCmd;

moveCmd.linear.x=0;

moveCmd.linear.y=0;

moveCmd.angular.z=0;

float diffx = targetPosex - robotX;

float diffy = targetPosey - robotY;

float yawDiff = normalizeRad(targetYaw - robotYaw);

int sign = yawDiff < 0?-1:1;

// check completed

if (fabs(diffx) < EPS && fabs(diffy) < EPS) {

if(fabs(yawDiff) < 2 \* EPS){

gotoPoseCompleted = true;

return moveCmd;

}else{

if(targetYaw == -100){

gotoPoseCompleted = true;

return moveCmd;

}

moveCmd.angular.z = yawDiff;

if(fabs(moveCmd.angular.z) > MAX\_TURN\_SPD){

moveCmd.angular.z=MAX\_TURN\_SPD \* sign;

}

if(fabs(moveCmd.angular.z) < MIN\_TURN\_SPD){

moveCmd.angular.z = MIN\_TURN\_SPD \* sign;

}

return moveCmd;

}

}

// if goto pose is not completed try to go to the position

if (!gotoPoseCompleted) {

targetYaw = atan2(diffy, diffx);

sign = yawDiff < 0?-1:1;

// if goto pose is not completed try to go to the position

yawDiff = normalizeRad(targetYaw - robotYaw);

sign = yawDiff < 0?-1:1;

moveCmd.angular.z = yawDiff;

if(fabs(moveCmd.angular.z) > MAX\_TURN\_SPD){

moveCmd.angular.z=MAX\_TURN\_SPD \* sign;

}

if(fabs(moveCmd.angular.z) < MIN\_TURN\_SPD){

moveCmd.angular.z = MIN\_TURN\_SPD \* sign;

}

if (fabs(yawDiff) < 0.2) {

float linearK = 0.7;

moveCmd.linear.x = linearK \* sqrt(pow(diffy,2) + pow(diffx,2));

if (moveCmd.linear.x > MAX\_LINEAR\_SPD){

moveCmd.linear.x = MAX\_LINEAR\_SPD;

}

}

}

return moveCmd;

}

void isbeavioractive\_func (const std\_msgs::String::ConstPtr& StringPointer ){

if(StringPointer->data == "go\_near\_beacon"){

isbeavioractive = true;

}else{

isbeavioractive = false;

behavior\_changed = true;

}

}

void pass\_func(const std\_msgs::Bool::ConstPtr& BoolPointer ){

if(BoolPointer->data ){

pass = true;

}else{

pass = false;

}

}

int main(int argc,char \*\*argv)//0.3

{

ros::init(argc,argv,"go\_near\_beacon");

ros::NodeHandle nh;

ros::Rate rate(30);

// read parameters

ros::NodeHandle private\_nh("~");

if (!private\_nh.getParam("prefix", prefix)){

}

ros::Subscriber BehaviorSubs = nh.subscribe( prefix + "curbehavior", 1, isbeavioractive\_func);

ros::Subscriber poseSubs = nh.subscribe( prefix + "amcl\_pose", 10, updatePose); //geometry\_msgs

ros::Publisher velPub=nh.advertise<geometry\_msgs::Twist>(prefix + "cmd\_vel",10);

completedPub = nh.advertise<std\_msgs::Bool>( prefix + "Red\_gotoPoseCompleted", 1);

ros::Publisher passPub = nh.advertise<std\_msgs::Bool>( "/Deliverpass", 1);

ros::Subscriber passSubs = nh.subscribe( "/Beaconpass", 1,pass\_func);

float targetx;

float targety ;

float targetyaw;

gotoPoseCompleted = true ;

robotposeGet = false ;

geometry\_msgs::Pose pose ;

std::vector<geometry\_msgs::Pose> init\_poses ;

pose.position.x = - 0.025 ;

pose.position.y = 0.57 ;

init\_poses.push\_back (pose) ;

pose.position.x = - 1.45 ;

pose.position.y = 0.57 ;

init\_poses.push\_back (pose) ;

pose.position.x = - 1.47;

pose.position.y = 1.68 ;

init\_poses.push\_back (pose) ;

std::vector<geometry\_msgs::Pose> poses ;

poses = init\_poses ;

bool go\_back = false;

behavior\_changed = true;

bool init = true;

pass = false;

while(ros::ok()){

if(robotposeGet){

if(!behavior\_changed){//reset values

poses = init\_poses ;

go\_back = true;

gotoPoseCompleted = true;

pass = false;

}

if(isbeavioractive && behavior\_changed){

if(go\_back){

geometry\_msgs::Twist moveCmd;

moveCmd.linear.x=-10;

moveCmd.linear.y=0;

moveCmd.angular.z=0;

velPub.publish(moveCmd);

usleep(900000);

go\_back = false;

}else{

if(gotoPoseCompleted){

targetx = robotX ;

targety = robotY ;

targetyaw = -100;

if(poses.size() == 0){

std\_msgs::Bool data;

data.data = true;

completedPub.publish(data);

behavior\_changed = false;

gotoPoseCompleted = false ;

}

if (!init){

if (poses.size() == 3){

//give signal

if(!pass){

std::cout << prefix + " waiting for pass" << std::endl;

continue;}

else{pass= false;}

}

}

if (poses.size() == 2){

//give signal

std\_msgs::Bool data;

data.data =true;

passPub.publish(data);

std::cout << prefix + " send pass" << std::endl;

}

if (poses.size() == 1) {

pose = poses.front();

std::cout <<prefix << "sending command pose.x=" << pose.position.x << " pose.y=" << pose.position.y << std::endl;

gotoPoseCompleted = false;

// remove the first element

poses.erase(poses.begin());

targetx = pose.position.x ;

targety = pose.position.y ;

targetyaw = PI/2;

init = false;

}

// if goto pose completed we can send new goto pose command

if (poses.size() > 1) {

pose = poses.front();

std::cout << prefix << "sending command pose.x=" << pose.position.x << " pose.y=" << pose.position.y << std::endl;

gotoPoseCompleted = false;

// remove the first element

poses.erase(poses.begin());

targetx = pose.position.x ;

targety = pose.position.y ;

targetyaw = -100 ;

}

}

velPub.publish(gotoPose(targetx,targety,targetyaw));

}

}

}

ros::spinOnce();

rate.sleep();

}

}

Formun Üstü

Formun Altı

**Order**

Uses: None

Common node for two robots , take order from conveyor band

Subscriptions:

* OmniPlatform001/curbehavior :for communication with my\_subsumption
* /OmniPlatform002/curbehavior : for communication with my\_subsumption
* /OmniPlatform001/amcl\_pose : currrent position of the OmniPlatform001

Publications:

* /order : order from conveyor band
* /OmniPlatform001/took\_order :for communication with my\_subsumption
* /OmniPlatform002/took\_order : for communication with my\_subsumption

#include <ros/ros.h>

#include <std\_msgs/Bool.h>

#include <std\_msgs/String.h>

#include <std\_msgs/Float32.h>

#include <sensor\_msgs/LaserScan.h>

#include <geometry\_msgs/Twist.h>

#include <geometry\_msgs/Pose.h>

#include <geometry\_msgs/PoseWithCovarianceStamped.h>

#include <geometry\_msgs/PoseWithCovariance.h>

#include <tf/transform\_listener.h>

#include <tf/tf.h>

#define PI 3.1415926535897932384626433f

#define MAX\_TURN\_SPD 0.52359879

#define MIN\_TURN\_SPD 0.05

#define MAX\_LINEAR\_SPD 0.7

#define EPS 0.05

bool isbeavioractive ;

bool behavior\_changed ;

float robotX;

float robotY;

float robotYaw;

std::string prefix;

int order\_num ;

void updatePose(const geometry\_msgs::PoseWithCovarianceStamped& NewPose){

geometry\_msgs::PoseWithCovariance RobotPosewithcovariance = NewPose.pose;

geometry\_msgs::Pose RobotPose = RobotPosewithcovariance.pose;

robotX=RobotPose.position.x;

robotY=RobotPose.position.y;

robotYaw=tf::getYaw(RobotPose.orientation);

}

float normalizeRad(float rad)

{

while(rad>PI)

{

rad=rad-2\*PI;

}

while(rad<-PI)

{

rad=rad+2\*PI;

}

return rad;

}

geometry\_msgs::Twist gotoPose(float targetPosex , float targetPosey , float targetYaw){

// create velocity vector

geometry\_msgs::Twist moveCmd;

moveCmd.linear.x=0;

moveCmd.linear.y=0;

moveCmd.angular.z=0;

// check completed

float diffx = targetPosex - robotX;

float diffy = targetPosey - robotY;

float yawDiff = normalizeRad(targetYaw - robotYaw);

if (fabs(diffx) < EPS && fabs(diffy) < EPS && fabs(yawDiff) < EPS){

return moveCmd;

}

// if goto pose is not completed try to go to the position

if(targetYaw = -100){

targetYaw = atan2(diffy, diffx);

}

int sign = yawDiff < 0?-1:1;

moveCmd.angular.z = yawDiff;

if(fabs(moveCmd.angular.z) > MAX\_TURN\_SPD)

{

moveCmd.angular.z=MAX\_TURN\_SPD \* sign;

}

if(fabs(moveCmd.angular.z) < MIN\_TURN\_SPD)

{

moveCmd.angular.z = MIN\_TURN\_SPD \* sign;

}

if (fabs(yawDiff) < 0.2) {

float linearK = 0.7;

moveCmd.linear.x = linearK \* sqrt(pow(diffy,2) + pow(diffx,2));

if (moveCmd.linear.x > MAX\_LINEAR\_SPD)

{

moveCmd.linear.x = MAX\_LINEAR\_SPD;

}

}

return moveCmd;

}

void isbeavioractive\_func1 (const std\_msgs::String::ConstPtr& StringPointer ){

if(StringPointer->data == "order"){

isbeavioractive = true;

prefix = "1" ;

}else if(prefix == "1"){

isbeavioractive = false;

behavior\_changed = true ;

}

}

void isbeavioractive\_func2 (const std\_msgs::String::ConstPtr& StringPointer ){

if(StringPointer->data == "order"){

isbeavioractive = true;

prefix = "2" ;

}else if(prefix == "2"){

isbeavioractive = false;

behavior\_changed = true ;

}

}

int main(int argc,char \*\*argv)

{

ros::init(argc,argv,"order");

ros::NodeHandle nh;

ros::Rate rate(30);

ros::Subscriber BehaviorSubs1 = nh.subscribe("/OmniPlatform001/curbehavior", 1, isbeavioractive\_func1);

ros::Subscriber BehaviorSubs2 = nh.subscribe("/OmniPlatform002/curbehavior", 1, isbeavioractive\_func2);

ros::Subscriber poseSubs = nh.subscribe( "/OmniPlatform001/amcl\_pose", 10, updatePose);

ros::Publisher orderPub =nh.advertise<std\_msgs::Float32>("/order",1);

ros::Publisher took\_orderPub1 =nh.advertise<std\_msgs::Bool>("/OmniPlatform001/took\_order",1);

ros::Publisher took\_orderPub2 =nh.advertise<std\_msgs::Bool>("/OmniPlatform002/took\_order",1);

float targetx;

float targety ;

float targetyaw;

float order = 1;

//order += 1;

behavior\_changed = true ;

while(ros::ok())

{

if(isbeavioractive && behavior\_changed){

behavior\_changed = false;

std\_msgs::Float32 ordernum;

ordernum.data = order;

orderPub.publish(ordernum);

usleep(3000000);

order = order + 1;

std\_msgs::Bool data;

data.data = true ;

if(prefix == "1"){took\_orderPub1.publish(data);}

else if(prefix == "2"){took\_orderPub2.publish(data);}

std::cout << "took\_order " << order -1 << std::endl;

}

ros::spinOnce();

rate.sleep();

}

}

Formun Üstü

Formun Altı

**Deliver order**

Uses: scan for localization

subscriptions :

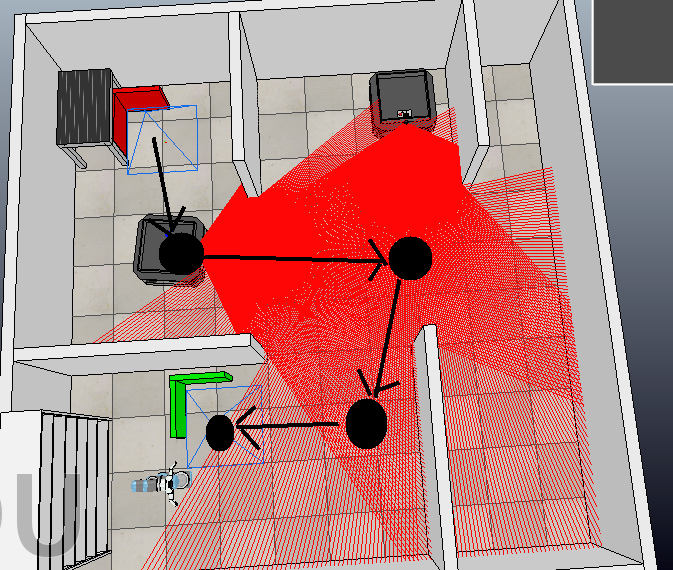
* /curbehavior with my\_subssumption node
* /amcl\_pose for localization
* Deliverpass with Go to red beacon node of other robot

publications :

* Beaconpass with deliver\_order node of other robot
* Green\_gotoPoseCompleted with my\_subssumption node
* /cmd\_vel for movement

Beaconpass and Deliverpass are the hierarchical waiting commands for multi agent system.

It goes to target via predetermined locations (black circles)



It waits for the delivery node in middle circle.

#include <ros/ros.h>

#include <std\_msgs/Bool.h>

#include <std\_msgs/String.h>

#include <sensor\_msgs/LaserScan.h>

#include <geometry\_msgs/Twist.h>

#include <geometry\_msgs/Pose.h>

#include <geometry\_msgs/Point.h>

#include <geometry\_msgs/PoseWithCovarianceStamped.h>

#include <geometry\_msgs/PoseWithCovariance.h>

#include <tf/transform\_listener.h>

#include <tf/tf.h>

#include <visualization\_msgs/MarkerArray.h>

#include <visualization\_msgs/Marker.h>

#include <nav\_msgs/OccupancyGrid.h>

#include <nav\_msgs/GetMap.h>

#include <move\_base\_msgs/MoveBaseAction.h>

#include <actionlib/client/simple\_action\_client.h>

#include <visualization\_msgs/MarkerArray.h>

#include <visualization\_msgs/Marker.h>

#include <nav\_msgs/OccupancyGrid.h>

#include <nav\_msgs/GetMap.h>

#include <std\_msgs/Bool.h>

#include <tf/transform\_listener.h>

#include <vector>

#define PI 3.1415926535897932384626433f

#define MAX\_TURN\_SPD 0.52359879

#define MIN\_TURN\_SPD 0.05

#define MAX\_LINEAR\_SPD 0.7

#define EPS 0.05

bool isbeavioractive ;

bool gotoPoseCompleted ;

bool robotposeGet ;

bool obstacle\_detected;

bool isrobot\_scan\_all\_area;

ros::Publisher completedPub;

float robotX;

float robotY;

float robotYaw;

std::string prefix;

float redx;

float redy;

float greenx;

float greeny;

bool took\_order;

bool behavior\_changed;

bool pass;

void updatePose(const geometry\_msgs::PoseWithCovarianceStamped& NewPose){

geometry\_msgs::PoseWithCovariance RobotPosewithcovariance = NewPose.pose;

geometry\_msgs::Pose RobotPose = RobotPosewithcovariance.pose;

robotX=RobotPose.position.x;

robotY=RobotPose.position.y;

robotYaw=tf::getYaw(RobotPose.orientation);

robotposeGet = true ;

}

float normalizeRad(float rad)

{

while(rad>PI)

{

rad=rad-2\*PI;

}

while(rad<-PI)

{

rad=rad+2\*PI;

}

return rad;

}

geometry\_msgs::Twist gotoPose(float targetPosex , float targetPosey , float targetYaw){

// create velocity vector

geometry\_msgs::Twist moveCmd;

moveCmd.linear.x=0;

moveCmd.linear.y=0;

moveCmd.angular.z=0;

float diffx = targetPosex - robotX;

float diffy = targetPosey - robotY;

float yawDiff = normalizeRad(targetYaw - robotYaw);

int sign = yawDiff < 0?-1:1;

// check completed

if (fabs(diffx) < EPS && fabs(diffy) < EPS) {

if(fabs(yawDiff) < 2 \* EPS){

gotoPoseCompleted = true;

return moveCmd;

}else{

if(targetYaw == -100){

gotoPoseCompleted = true;

return moveCmd;

}

moveCmd.angular.z = yawDiff;

if(fabs(moveCmd.angular.z) > MAX\_TURN\_SPD){

moveCmd.angular.z=MAX\_TURN\_SPD \* sign;

}

if(fabs(moveCmd.angular.z) < MIN\_TURN\_SPD){

moveCmd.angular.z = MIN\_TURN\_SPD \* sign;

}

return moveCmd;

}

}

// if goto pose is not completed try to go to the position

if (!gotoPoseCompleted) {

targetYaw = atan2(diffy, diffx);

sign = yawDiff < 0?-1:1;

// if goto pose is not completed try to go to the position

yawDiff = normalizeRad(targetYaw - robotYaw);

sign = yawDiff < 0?-1:1;

moveCmd.angular.z = yawDiff;

if(fabs(moveCmd.angular.z) > MAX\_TURN\_SPD){

moveCmd.angular.z=MAX\_TURN\_SPD \* sign;

}

if(fabs(moveCmd.angular.z) < MIN\_TURN\_SPD){

moveCmd.angular.z = MIN\_TURN\_SPD \* sign;

}

if (fabs(yawDiff) < 0.2) {

float linearK = 0.7;

moveCmd.linear.x = linearK \* sqrt(pow(diffy,2) + pow(diffx,2));

if (moveCmd.linear.x > MAX\_LINEAR\_SPD){

moveCmd.linear.x = MAX\_LINEAR\_SPD;

}

}

}

return moveCmd;

}

void isbeavioractive\_func (const std\_msgs::String::ConstPtr& StringPointer ){

if(StringPointer->data == "deliver\_order"){

if(!isbeavioractive){pass = false;}

isbeavioractive = true;

}else{

isbeavioractive = false;

behavior\_changed = true;

}

}

void pass\_func(const std\_msgs::Bool::ConstPtr& BoolPointer ){

if(BoolPointer->data ){

pass = true;

}else{

pass = false;

}

}

int main(int argc,char \*\*argv)//0.3

{

ros::init(argc,argv,"go\_near\_beacon");

ros::NodeHandle nh;

ros::Rate rate(30);

// read parameters

// read parameters

ros::NodeHandle private\_nh("~");

if (!private\_nh.getParam("prefix", prefix)){

//initialx = 0;

}

ros::Subscriber BehaviorSubs = nh.subscribe(prefix + "curbehavior", 1, isbeavioractive\_func);

ros::Subscriber poseSubs = nh.subscribe(prefix + "amcl\_pose", 10, updatePose); //geometry\_msgs

ros::Publisher velPub=nh.advertise<geometry\_msgs::Twist>(prefix + "cmd\_vel",10);

completedPub = nh.advertise<std\_msgs::Bool>( prefix + "Green\_gotoPoseCompleted", 1);

ros::Subscriber passSubs = nh.subscribe( "/Deliverpass", 1,pass\_func);

ros::Publisher passPub = nh.advertise<std\_msgs::Bool>( "/Beaconpass", 1);

float targetx;

float targety ;

float targetyaw;

bool go\_back = true;

gotoPoseCompleted = true ;

robotposeGet = false ;

geometry\_msgs::Pose pose ;

std::vector<geometry\_msgs::Pose> init\_poses ;

std::vector<geometry\_msgs::Pose> poses ;

pose.position.x = - 1.50 ;

pose.position.y = 0.57 ;

init\_poses.push\_back (pose) ;

pose.position.x = +9.2500e-01 ;

pose.position.y = +5.2290e-01 ;

init\_poses.push\_back (pose) ;

pose.position.x = +5.0000e-02 ;

pose.position.y = -1.1550e+00 ;

init\_poses.push\_back (pose) ;

pose.position.x = -0.85 ;

pose.position.y = -1.18 ;

init\_poses.push\_back (pose) ;

//redx -0.121041redy 0.790346

//greenx -1.40969greeny -0.319107

//redx = -1.84851 + 0.25 ;

//redy = 0.790346 + 0.25 ;

//greenx = 2 ;//-1.40969 + 0.25;

//greeny = 0;//-0.319107 + 0.25;

poses = init\_poses;

behavior\_changed = true;

pass = false;

while(ros::ok()){

if(robotposeGet){

if(!behavior\_changed){

go\_back = true;

gotoPoseCompleted = true ;

poses = init\_poses;

pass = false;

}

if(isbeavioractive && behavior\_changed){

if(go\_back){

geometry\_msgs::Twist moveCmd;

moveCmd.linear.x=-10;

moveCmd.linear.y=0;

moveCmd.angular.z=0;

velPub.publish(moveCmd);

usleep(1000000);

go\_back = false;

}else{

if(gotoPoseCompleted){

targetx = robotX ;

targety = robotY ;

targetyaw = -100;

if (poses.size() == 2){

//wait for signal

std\_msgs::Bool data;

data.data =true;

passPub.publish(data);

if(!pass){

//std::cout << prefix + "waiting for pass" << std::endl;

continue;}

else{

std::cout << prefix + "pass received" << std::endl;

pass= false;}

}

if(poses.size() == 0){

std\_msgs::Bool data;

data.data = true;

completedPub.publish(data);

std::cout << prefix + "sending pass" << std::endl;

behavior\_changed = false;

gotoPoseCompleted = false ;

}

if (poses.size() == 1) {

pose = poses.front();

std::cout <<prefix << "sending command pose.x=" << pose.position.x << " pose.y=" << pose.position.y << std::endl;

gotoPoseCompleted = false;

// remove the first element

poses.erase(poses.begin());

targetx = pose.position.x ;

targety = pose.position.y ;

targetyaw = PI;

}

// if goto pose completed we can send new goto pose command

if (poses.size() > 1) {

pose = poses.front();

std::cout << prefix << "sending command pose.x=" << pose.position.x << " pose.y=" << pose.position.y << std::endl;

gotoPoseCompleted = false;

// remove the first element

poses.erase(poses.begin());

targetx = pose.position.x ;

targety = pose.position.y ;

targetyaw = -100 ;

}

}

velPub.publish(gotoPose(targetx,targety,targetyaw));

}

}

}

ros::spinOnce();

rate.sleep();

}

}

Formun Üstü

Formun Altı

**Pick up order**

Uses: GrenBeaconDepth for locazlization of object and GrenBeaconRGB for color

#include <ros/ros.h>

#include <std\_msgs/Bool.h>

#include <std\_msgs/String.h>

#include <sensor\_msgs/LaserScan.h>

#include <geometry\_msgs/Twist.h>

#include <geometry\_msgs/Pose.h>

#include <geometry\_msgs/PoseWithCovarianceStamped.h>

#include <geometry\_msgs/PoseWithCovariance.h>

#include <tf/transform\_listener.h>

#include <tf/tf.h>

#include "std\_msgs/Float32.h"

#include "geometry\_msgs/PoseStamped.h"

#include "geometry\_msgs/PoseArray.h"

#include <sensor\_msgs/PointCloud2.h>

#include "kdl/chain.hpp"

#include "kdl/chainfksolver.hpp"

#include "kdl/chainfksolverpos\_recursive.hpp"

#include "kdl/chainiksolverpos\_lma.hpp"

#include "kdl/frames\_io.hpp"

#include "image\_helper.h"

#include <pcl\_ros/transforms.h>

#include <tf/transform\_listener.h>

#include <pcl/visualization/pcl\_visualizer.h>

#include <pcl/sample\_consensus/method\_types.h>

#include <pcl/sample\_consensus/model\_types.h>

#include <pcl/segmentation/sac\_segmentation.h>

#include <pcl/segmentation/sac\_segmentation.h>

#include <pcl/filters/extract\_indices.h>

#include <pcl\_ros/point\_cloud.h>

#define UR10\_DOF 6

#define PI 3.1415926535897932384626433f

#define MAX\_TURN\_SPD 0.52359879

#define MIN\_TURN\_SPD 0.05

#define MAX\_LINEAR\_SPD 0.7

#define EPS 0.05

ros::Subscriber cloudSubs;

ros::Subscriber ballPoseSubs;

geometry\_msgs::PoseStamped endEffectorPosition;

geometry\_msgs::PoseArray cupPositions;

geometry\_msgs::Pose calculatedPosition;

geometry\_msgs::PoseArray cupTargetPositions;

std::string prefix;

ImageHelper imageHelper;

bool isCupTargetPositionsAvailable = false;

bool is\_cup\_taken = false; //

bool shelf\_reached = false;

bool isCupPositionsAvailable ;

bool isbeavioractive ;

bool behavior\_changed ;

float robotX;

float robotY;

float robotYaw;

float objectx;

float objecty;

float objectz;

float shelfx;

float shelfy;

float shelfz;

bool error\_reset = true;

ros::Publisher beaconEnable ;

float calculateError (const geometry\_msgs::PoseStamped& endEffectorPosition) {

float error = 0;

error += pow(calculatedPosition.position.x - endEffectorPosition.pose.position.x,2) \* 10000;

error += pow(calculatedPosition.position.y - endEffectorPosition.pose.position.y,2) \* 10000;

error += pow(calculatedPosition.position.z - endEffectorPosition.pose.position.z,2) \* 10000;

error = sqrt(error);

if(error < 8 && error\_reset) {

if(shelf\_reached){behavior\_changed = false;

calculatedPosition.position.x = 0;

calculatedPosition.position.y = 0;

calculatedPosition.position.z = 0;}

if(is\_cup\_taken){shelf\_reached = true; std::cout << "1" << std::endl ;}

is\_cup\_taken = true; //

error\_reset = false;

std::cout << "cup is taken" << std::endl ;

} else{

//std::cout << "error " << error << std::endl;

}

return error;

}

void endEffectorPositionCallback(const geometry\_msgs::PoseStampedConstPtr& endEffectorPositionMessage) {

endEffectorPosition = \*endEffectorPositionMessage;

//std::cout << "Calculated Position :" << "[ "<< calculatedPosition.position.x << " , " << calculatedPosition.position.y << " , " << calculatedPosition.position.z << " ]" << std::endl;

// std::cout << "End Effector Position :" << "[ "<< endEffectorPositionMessage->pose.position.x << " , " << endEffectorPositionMessage->pose.position.y << " , " << endEffectorPositionMessage->pose.position.z << " ]" << std::endl;

calculateError(\*endEffectorPositionMessage);

}

KDL::Chain initChainUR10() {

/\*Base Position :[ -1.49143 , -1.59999 , 0.0446423 ]

joint positions are relative to base

Joint1 Position :[ -0.0926735 , 1.07288e-06 , 0.083327 ]

Joint2 Position :[ -0.112951 , 0.000450611 , 0.695387 ]

Joint3 Position :[ -0.107075 , -0.000293493 , 1.26759 ]

Joint4 Position :[ -0.165607 , -0.000283599 , 1.32478 ]

Joint5 Position :[ -0.223011 , -0.000267982 , 1.38309 ]

End Effector Position :[ -0.333581 , -0.000296116 , 1.38248 ]

\*/

KDL::Chain chain;

KDL::Segment s0 = KDL::Segment(KDL::Joint(KDL::Joint::RotZ), //z

KDL::Frame(KDL::Rotation::RPY(0.0, 0.0 ,PI),

KDL::Vector(-0.0926735 , -1.07288e-06 , 0.083327) ) //ok

);

KDL::Segment s1 = KDL::Segment(KDL::Joint(KDL::Joint::RotX), //-x

KDL::Frame(KDL::Rotation::RPY(0.0,0.0,0.0),

KDL::Vector(-0.0926735 - (-0.112951) ,1.07288e-06 - (0.000450611) , 0.695387 - 0.083327) )

);

KDL::Segment s2 = KDL::Segment(KDL::Joint(KDL::Joint::RotX), //-x

KDL::Frame(KDL::Rotation::RPY(0.0,0.0,0.0),

KDL::Vector(-0.112951 - (-0.107075),0.000450611 - (-0.000293493),1.26759-0.695387 ) )

);

KDL::Segment s3 = KDL::Segment(KDL::Joint(KDL::Joint::RotX), //-x

KDL::Frame(KDL::Rotation::RPY(0.0,0.0,0.0),

KDL::Vector(-0.107075 - (-0.165607),-0.000293493 -(-0.000283599 ),1.32478-1.26759) )

);

KDL::Segment s4 = KDL::Segment(KDL::Joint(KDL::Joint::RotZ), //z

KDL::Frame(KDL::Rotation::RPY(0.0,0.0,0.0),

KDL::Vector(-0.165607 - (-0.223011), -0.000283599 - (-0.000267982),1.38309-1.32478) )

);

KDL::Segment s5 = KDL::Segment(KDL::Joint(KDL::Joint::RotX), //-x

KDL::Frame(KDL::Rotation::RPY(0.0,0.0,0.0),

KDL::Vector(-0.223011 - (-0.333581) ,-0.000267982 - (-0.000296116) ,1.38248-1.38309) )

);

chain.addSegment(s0);

chain.addSegment(s1);

chain.addSegment(s2);

chain.addSegment(s3);

chain.addSegment(s4);

chain.addSegment(s5);

return chain;

}

void updatePose(const geometry\_msgs::PoseWithCovarianceStamped& NewPose){

geometry\_msgs::PoseWithCovariance RobotPosewithcovariance = NewPose.pose;

geometry\_msgs::Pose RobotPose = RobotPosewithcovariance.pose;

robotX=RobotPose.position.x;

robotY=RobotPose.position.y;

robotYaw=tf::getYaw(RobotPose.orientation);

}

float normalizeRad(float rad)

{

while(rad>PI)

{

rad=rad-2\*PI;

}

while(rad<-PI)

{

rad=rad+2\*PI;

}

return rad;

}

void isbeavioractive\_func1 (const std\_msgs::String::ConstPtr& StringPointer ){

if(StringPointer->data == "pick\_up\_order"){

isbeavioractive = true;

std\_msgs::Bool data;

data.data = true;

beaconEnable.publish(data);

prefix = "1" ;

}else if(prefix == "1"){

isbeavioractive = false;

behavior\_changed = true;

}

//std::cout <<"omni1" << prefix << std::endl;

}

void isbeavioractive\_func2 (const std\_msgs::String::ConstPtr& StringPointer ){

if(StringPointer->data == "pick\_up\_order"){

isbeavioractive = true;

std\_msgs::Bool data;

data.data = true;

beaconEnable.publish(data);

prefix = "2" ;

}else if(prefix == "2"){

isbeavioractive = false;

behavior\_changed = true;

}

//std::cout << "omni1" << prefix << std::endl;

}

class StateEstimation {

ros::Subscriber cloudSubs;

ros::Subscriber ballPoseSubs;

ros::NodeHandle nh;

tf::TransformListener tfListener;

bool isAdded;

//pcl::visualization::PCLVisualizer\* viewer;

public:

StateEstimation() {

cloudSubs = nh.subscribe("/tools/sensors/greenBeaconDepth", 1,

&StateEstimation::pointCloudCallback, this);

isAdded = false;

//viewer = new pcl::visualization::PCLVisualizer("3D Viewer");

// viewer->initCameraParameters();

// viewer->setBackgroundColor (0, 0, 0);

// viewer->addCoordinateSystem (1.0);

}

void pointCloudCallback(const sensor\_msgs::PointCloud2ConstPtr& msg) {

if(!isCupPositionsAvailable){

ros::Time t = ros::Time(0);

// wait from transform

tfListener.waitForTransform("greenBeacon\_base", "/tools/sensors/greenBeaconDepth", t,

ros::Duration(1.0));

// convert point cloud frame to beacon frame

sensor\_msgs::PointCloud2 transformedPoints;

pcl\_ros::transformPointCloud("greenBeacon\_base", \*msg, transformedPoints,

tfListener);

//pcl::PointCloud<pcl::PointXYZ> \*pointCloud = new pcl::PointCloud<pcl::PointXYZ>();

pcl::PointCloud<pcl::PointXYZ>::Ptr pointCloud(new pcl::PointCloud<pcl::PointXYZ>());

pcl::fromROSMsg(transformedPoints, \*pointCloud);

std::cout << "size of pcl= " << pointCloud->points.size () << std::endl;

// DO: detect ball here

//plane detection

pcl::ModelCoefficients::Ptr coefficients (new pcl::ModelCoefficients);

pcl::PointIndices::Ptr inliers (new pcl::PointIndices);

// Create the segmentation object

pcl::SACSegmentation<pcl::PointXYZ> seg;

// Optional

seg.setOptimizeCoefficients (true);

// Mandatory

seg.setModelType (pcl::SACMODEL\_PLANE);

seg.setMethodType (pcl::SAC\_RANSAC);

seg.setDistanceThreshold (0.01);

seg.setInputCloud (pointCloud);

seg.segment (\*inliers, \*coefficients);

if (inliers->indices.size () == 0)

{

PCL\_ERROR ("Could not estimate a planar model for the given dataset.");

}

/\*std::cout << "Model coefficients: " << coefficients->values[0] << " "

<< coefficients->values[1] << " "

<< coefficients->values[2] << " "

<< coefficients->values[3] << std::endl;

std::cout << "Model inliers: " << inliers->indices.size () << std::endl; \*/

//for (size\_t i = 0; i < inliers->indices.size (); ++i)

// inliers->indices[i] << " " << pointCloud->points[inliers->indices[i]].x << " "

// << pointCloud->points[inliers->indices[i]].y << " "

// << pointCloud->points[inliers->indices[i]].z << std::endl;

//remove indices of plane

pcl::ExtractIndices<pcl::PointXYZ> extract;

extract.setInputCloud(pointCloud);

extract.setIndices(inliers);

extract.setNegative(true);

extract.filter(\*pointCloud);

std::cout << "size after plane removal = " << pointCloud->points.size () << std::endl;

//plane removed view

pcl::PointCloud<pcl::PointXYZ>::ConstPtr constPointCloud(pointCloud);

//viewer->removePointCloud("cloud");

//viewer->addPointCloud(constPointCloud, "cloud");

//viewer->setPointCloudRenderingProperties (pcl::visualization::PCL\_VISUALIZER\_POINT\_SIZE, 1, "cloud");

//viewer->spinOnce(100);

//limit x axis to detect only ball

int cloudsize= 0 ;

for (int i = 0; i < pointCloud->points.size (); i++) {

//cout << "x:" << pointCloud->points[i].x << "y:" << pointCloud->points[i].y << std::endl;

if( pointCloud->points[i].y-0.949998 < -1.09 and pointCloud->points[i].y-0.949998 > -1.2 and pointCloud->points[i].x-1.1 <-1.11){

}

else if( pointCloud->points[i].x-1.1 < -0.84 and pointCloud->points[i].x-1.1 >-1.12){

if( pointCloud->points[i].y-0.949998 < -0.91 and pointCloud->points[i].y-0.949998 > -1.3){

cout << "x:" << pointCloud->points[i].x << "y:" << pointCloud->points[i].y << std::endl;

cloudsize++ ;

}

}

}

//create new cloud data for ball

pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZ>);

// Fill in the cloud data

cloud->width = cloudsize;

cloud->height = 1;

cloud->points.resize (cloud->width \* cloud->height);

cloudsize= 0 ;

float measured\_x = 0;

float measured\_y = 0;

float measured\_z = 0;

for (int i = 0; i < pointCloud->points.size (); i++) {

if( pointCloud->points[i].y-0.949998 < -0.952 and pointCloud->points[i].y-0.949998 > -1.2 and pointCloud->points[i].x-1.1 <-1.11){

}

else if( pointCloud->points[i].x-1.1 < -0.84 and pointCloud->points[i].x-1.1 >-1.12){

if( pointCloud->points[i].y-0.949998 < -0.91 and pointCloud->points[i].y-0.949998 > -1.3){

cloud->points[cloudsize].x = pointCloud->points[i].x ;

measured\_x = measured\_x + pointCloud->points[i].x;

cloud->points[cloudsize].y = pointCloud->points[i].y ;

measured\_y = measured\_y + pointCloud->points[i].y ;

cloud->points[cloudsize].z = pointCloud->points[i].z ;

measured\_z = measured\_z + pointCloud->points[i].z;

cloudsize++ ;

}

}

}

if(cloudsize > 0 ){//greenbeacon Position :[ -1.1 , -0.949998 , 0.25 ]yaw 0

objectx = measured\_x/cloudsize ;

objecty = measured\_y/cloudsize ;

objectz = measured\_z/cloudsize ;

if(objectx > 0.38){//cylynder

std::cout << "cyl " << std::endl ;

shelfz = +8.5064e-01;

}else if(objecty < -0.33){

std::cout << "cyl " << std::endl ;

shelfz = +8.5064e-01;

}else{//cube

std::cout << "cube " << std::endl ;

shelfz = +1.2306e+00;

}//local x:0.244029 y:-0.164809 z:0.1055

//global x:0.494029 y:-1.11481 z:-0.99

//real -0.85189e -1.1299e+00 +0.33054

std::cout << "local" << " x:" << objectx << " y:" << objecty << " z:" << objectz ; //

objectz = 0.25 + objectz ; //

//float temp = objecty;

objecty = -0.949998 + objecty ;

objectx = -1.1 + objectx ;

isCupPositionsAvailable = true;

std::cout << "global" << " x:" << objectx << " y:" << objecty << " z:" << objectz ; //

// viewer->removePointCloud("cloud");

// viewer->addPointCloud(cloud, "cloud");

// viewer->setPointCloudRenderingProperties (pcl::visualization::PCL\_VISUALIZER\_POINT\_SIZE, 1, "cloud");

// viewer->spinOnce(100);

//Base Position :[ -1.49143 , -1.59999 , 0.0446423 ]yaw -3.14157

calculatedPosition.position.x = -1.4914e+00 - objectx;

calculatedPosition.position.y = -1.59999 - objecty;

calculatedPosition.position.z = objectz - 0.0446423 + 0.1;

}

}

}

};

int main(int argc,char \*\*argv)

{

std::cout << "PLC Start..." << std::endl;

ros::init(argc,argv,"pick\_up\_order");

ros::NodeHandle nh;

ros::Rate rate(30);

ros::Publisher joints[6];

joints[0] = nh.advertise<std\_msgs::Float32>("/UR10/jointAngles/joint001",1000);

joints[1] = nh.advertise<std\_msgs::Float32>("/UR10/jointAngles/joint002",1000);

joints[2] = nh.advertise<std\_msgs::Float32>("/UR10/jointAngles/joint003",1000);

joints[3] = nh.advertise<std\_msgs::Float32>("/UR10/jointAngles/joint004",1000);

joints[4] = nh.advertise<std\_msgs::Float32>("/UR10/jointAngles/joint005",1000);

joints[5] = nh.advertise<std\_msgs::Float32>("/UR10/jointAngles/joint006",1000);

for(int index=0;index<UR10\_DOF;index++){

std\_msgs::Float32 jointMessage;

jointMessage.data = 0;

joints[index].publish(jointMessage);

}

ros::Subscriber endEffectorPositionSubscriber = nh.subscribe<geometry\_msgs::PoseStamped>("/UR10/positions/endEffectorPosition",1000,endEffectorPositionCallback);

// ros::Subscriber cupPositionsSubscriber = nh.subscribe<geometry\_msgs::PoseArray>("/task/positions/cups",1000,cupPositionaCallback);

//ros::Subscriber cupTargetPositionsSubscriber = nh.subscribe<geometry\_msgs::PoseArray>("/task/positions/cuptargets",1000,cupTargetPositionaCallback);

ros::Publisher gripper = nh.advertise<std\_msgs::Float32>("/UR10/gripper/closingAngle",1000);

ros::Publisher PickedupPub1 =nh.advertise<std\_msgs::Bool>("/OmniPlatform001/Pickedupobject",1);

ros::Publisher PickedupPub2 =nh.advertise<std\_msgs::Bool>("/OmniPlatform002/Pickedupobject",1);

//ros::Rate loop(10);

KDL::Chain chain = initChainUR10();

geometry\_msgs::PoseArray goalPositions;

ros::Subscriber BehaviorSubs1 = nh.subscribe("/OmniPlatform001/curbehavior", 1, isbeavioractive\_func1);

ros::Subscriber BehaviorSubs2 = nh.subscribe("/OmniPlatform002/curbehavior", 1, isbeavioractive\_func2);

ros::Subscriber imageSubs = nh.subscribe("/tools/sensors/greenBeaconRGB", 10, &ImageHelper::processImage, &imageHelper);

ros::Subscriber poseSubs = nh.subscribe( "/OmniPlatform001/amcl\_pose", 10, updatePose); //geometry\_msgs

beaconEnable = nh.advertise<std\_msgs::Bool>("/tools/sensors/greenBeaconEnable", 1);

StateEstimation stateEstimation;

int gripper\_count = 0 ;

bool cup\_gripped = false;

std\_msgs::Bool data;

data.data = true;

behavior\_changed = true;

isCupPositionsAvailable = false;

shelfx = -1.95;

while(ros::ok()){

if(!behavior\_changed){//reset values

shelf\_reached = false;

gripper\_count = 0;

isCupPositionsAvailable = false;

isCupTargetPositionsAvailable = false ;

is\_cup\_taken =false;

cup\_gripped = false;

error\_reset = true;

prefix = " " ;

}

if(shelf\_reached){

if(prefix == "1"){PickedupPub1.publish(data);}

else if(prefix == "2"){PickedupPub2.publish(data);}

behavior\_changed =false ;

std\_msgs::Float32 jointMessage;

KDL::ChainFkSolverPos\_recursive forwardKinematicSolver = KDL::ChainFkSolverPos\_recursive(chain);

KDL::Frame goalEndEffectorPosition;

jointMessage.data = 0;

gripper.publish(jointMessage);

calculatedPosition.position.x = -0.333581 ;

calculatedPosition.position.y = -0.000296116;

calculatedPosition.position.z = 1.38248 ;

goalEndEffectorPosition = KDL::Frame(

KDL::Rotation::RPY(PI/2,0,PI),

KDL::Vector(calculatedPosition.position.x,calculatedPosition.position.y,calculatedPosition.position.z));

// You can use this matrix for ChainIkSolverPos\_LMA

Eigen::Matrix<double,6,1> L;

L(0)=1;L(1)=1;L(2)=1;

L(3)=0.01;L(4)=0.01;L(5)=0.01;

//create iksolver

KDL::ChainIkSolverPos\_LMA iksolver = KDL::ChainIkSolverPos\_LMA(chain,L );

// Create joint arrays

unsigned int nj = chain.getNrOfJoints();

KDL::JntArray initialjointpositions = KDL::JntArray(nj);

KDL::JntArray jointpositions = KDL::JntArray(nj);

int ret = iksolver.CartToJnt(initialjointpositions, goalEndEffectorPosition, jointpositions);

// Create the frame that will contain the results

KDL::Frame cartpos;

// Calculate forward position kinematics

bool kinematics\_status;

kinematics\_status = forwardKinematicSolver.JntToCart(jointpositions,cartpos);

if(kinematics\_status>=0){

}else{

printf("%s \n","Error: could not calculate forward kinematics :(");

}

/\*

\* DO: Publish the found joint angles.

\*/

if(ret == 0) {

jointMessage.data = jointpositions(0);

joints[0].publish(jointMessage);

jointMessage.data = jointpositions(1);

joints[1].publish(jointMessage);

jointMessage.data = jointpositions(2);

joints[2].publish(jointMessage);

jointMessage.data = jointpositions(3);

joints[3].publish(jointMessage);

jointMessage.data = jointpositions(4);

joints[4].publish(jointMessage);

jointMessage.data = jointpositions(5);

joints[5].publish(jointMessage);

}else{

std::cout << ret ; printf("%s \n","Error: could not calculate inverse kinematics :(");

}

error\_reset = true ;

ros::spinOnce();

rate.sleep();

continue;

}

if(isbeavioractive && behavior\_changed){

//findcuptarget pos

if(isCupPositionsAvailable){

if(imageHelper.isbluefound){

//std::cout << "blue " << std::endl ;

shelfy = -1.9046e+00;

}else if(imageHelper.isgreenfound){

//std::cout << "green " << std::endl ;

shelfy = -1.2546e+00;

}

isCupTargetPositionsAvailable = true;

}

if(isCupPositionsAvailable && isCupTargetPositionsAvailable) {

KDL::ChainFkSolverPos\_recursive forwardKinematicSolver = KDL::ChainFkSolverPos\_recursive(chain);

KDL::Frame goalEndEffectorPosition;

std\_msgs::Float32 jointMessage;

jointMessage.data = 5.5;

if(is\_cup\_taken) {

gripper.publish(jointMessage); gripper\_count++ ;

if(gripper\_count > 15) {

cup\_gripped = true;

//publish cup taken

}

}

if (cup\_gripped){ //Base Position :[ -1.49143 , -1.59999 , 0.0446423 ]yaw -3.14157

calculatedPosition.position.x = -1.4914e+00 - shelfx ;

calculatedPosition.position.y = -1.59999 - shelfy;

calculatedPosition.position.z = shelfz - +0.0446423;

}

if(cup\_gripped){

goalEndEffectorPosition = KDL::Frame(

KDL::Rotation::RPY(0,0,0),

KDL::Vector(calculatedPosition.position.x,calculatedPosition.position.y,calculatedPosition.position.z));

error\_reset = true ;

//-0.714789 , -0.155105 , 0.88193)

}else{ goalEndEffectorPosition = KDL::Frame(

KDL::Rotation::RPY(0,PI/2,0),

KDL::Vector( calculatedPosition.position.x,calculatedPosition.position.y,calculatedPosition.position.z));

}

// You can use this matrix for ChainIkSolverPos\_LMA

Eigen::Matrix<double,6,1> L;

L(0)=1;L(1)=1;L(2)=1;

L(3)=0.01;L(4)=0.01;L(5)=0.01;

//create iksolver

KDL::ChainIkSolverPos\_LMA iksolver = KDL::ChainIkSolverPos\_LMA(chain,L );

// Create joint arrays

unsigned int nj = chain.getNrOfJoints();

KDL::JntArray initialjointpositions = KDL::JntArray(nj);

KDL::JntArray jointpositions = KDL::JntArray(nj);

int ret = iksolver.CartToJnt(initialjointpositions, goalEndEffectorPosition, jointpositions);

// Create the frame that will contain the results

KDL::Frame cartpos;

// Calculate forward position kinematics

bool kinematics\_status;

kinematics\_status = forwardKinematicSolver.JntToCart(jointpositions,cartpos);

if(kinematics\_status>=0){

//cartpos.M rot matrix

//cartpos.p end effector pos

//std::cout << "KDL end effector :"<< cartpos.p <<std::endl;

//printf("%s \n","Succes");

}else{

printf("%s \n","Error: could not calculate forward kinematics :(");

}

/\*

\* DO: Publish the found joint angles.

\*/

if(ret == 0) {

jointMessage.data = jointpositions(0);

joints[0].publish(jointMessage);

jointMessage.data = jointpositions(1);

joints[1].publish(jointMessage);

jointMessage.data = jointpositions(2);

joints[2].publish(jointMessage);

jointMessage.data = jointpositions(3);

joints[3].publish(jointMessage);

jointMessage.data = jointpositions(4);

joints[4].publish(jointMessage);

jointMessage.data = jointpositions(5);

joints[5].publish(jointMessage);}

else{ std::cout << ret ; printf("%s \n","Error: could not calculate inverse kinematics :("); calculatedPosition.position.z = calculatedPosition.position.z + 0.01; }

}

}

ros::spinOnce();

rate.sleep();

}

}