**−Table of Contents**

* [Introduction](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment9#introduction)
* [Video(Time stamps are in the description)](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment9#video_time_stamps_are_in_the_description)
* [Code](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment9#code)
* [Deliverables](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment9" \l "deliverables)

**Introduction**

In this assignment we implemented a subsumption algorithm with 3 levels. Our goal is finding the red ball in the unknown map. We use laser scan, gmapping and rgb camera on the hokuyo robot.

Yo make the search I coded a general wall following robot , so algorithm doesn't need to be changed for a new map.

From top to bottom:

* Obstacle Avoidance

If encountered with obstacle ,turn 90 degrees to the right .

* Go to Ball

If you see the ball turn to make ball stay in middle and go towards to ball.

* Search the ball

Follow wall until it ends. If it ends, turn 90 degrees to the left and go straight until wall is detected. Rinse and repeat.

As all the turns are all 90 degrees , robot only moves orthogonally. This make the solution and coding simpler. It turns left via **Search the Ball** and turns right via **Obstacle avoidance** .

To make it generally robot starts with going straight until it hits a obstacle, then **Obstacle avoidance** turns it to the right and that means now there is a wall on its left. Then it stats to follow that wall until :   
*a) It detects a obstacle ;*   
In this case robot turns right and now there is a wall on its left again.  
*b) Followed wall ends;*   
In this case robot turns left and goes straight until it detects the walls end point. Now there is another wall on its left to follow.

Formun Üstü

Formun Altı

**Video(Time stamps are in the description)**

[**https://youtu.be/w7CfOnJ9maE**](https://youtu.be/w7CfOnJ9maE)

Formun Üstü

Formun Altı

**Code**

I used my gmapping variables in the assignment 7. I needed to add (double) before myImage(i,j)[0] to get the 0-255 red data.

Formun Üstü

Formun Altı

**Map Helper**

**map\_helper.cpp processMap:** Update the occupancygrid.

void MapHelper::processMap(const nav\_msgs::OccupancyGridConstPtr& occupancyGridConstPtr)

{

resolution = occupancyGridConstPtr->info.resolution;

width = occupancyGridConstPtr->info.width;

height = occupancyGridConstPtr->info.height;

originx = occupancyGridConstPtr->info.origin.position.x;

originy = occupancyGridConstPtr->info.origin.position.y;

occupancyGrid = \*occupancyGridConstPtr;

}

Formun Üstü

Formun Altı

**General Sensor Helper**

**general\_sensor\_helper.h:** Added new variables.

* int direction ; can be 1,2,3,4 depending of the facing direction
* bool isoccupiedleft; Is left of the robot is occupied
* bool start; becomes true after first obstacle detection

**general\_sensor\_helper.cpp:** Changed the obstacleThreshold = 0.5; and made +-30 scan instead of +-20

GeneralSensorHelper::GeneralSensorHelper() {

isObstacle = false;

direction = 10 ; //modifieds

isoccupiedleft = false;

start = false ;

}

void GeneralSensorHelper::setRobotTransform(tf::StampedTransform& \_robotTransform)

{

robotTransform = \_robotTransform;

//determine direction

double anglethreshold\_1 = 0.2; //2 degrees

double anglethreshold\_2\_3 = 0.2; //2 degrees

double anglethreshold\_0 = 0.35; //2 degrees

double angle = tf::getYaw(robotTransform.getRotation()) ;

// 0 +x 0

//3 +y PI/2 1 -y -PI/2

// 2 -x PI

if(angle > 0-anglethreshold\_0 && angle < 0+anglethreshold\_0){direction = 0;}

else if (angle > -PI/2-anglethreshold\_1 && angle < -PI/2+anglethreshold\_1 ){direction = 1;}

else if (angle > PI/2-anglethreshold\_2\_3 && angle < PI/2 + anglethreshold\_2\_3){direction = 3;}

else if (angle < -PI+anglethreshold\_2\_3 || angle > PI-anglethreshold\_2\_3){direction = 2;}

std::cout << "Orientation is " << direction << " angle is " << angle << std::endl;

if(start){

double x = robotTransform.getOrigin().x();

double y = robotTransform.getOrigin().y();

if(direction == 0){isoccupiedleft = mapHelper.isOccupied(x-0.5,y+1); }

else if(direction == 3){isoccupiedleft = mapHelper.isOccupied(x-1,y-0.5); }

else if(direction == 1){isoccupiedleft = mapHelper.isOccupied(x+1,y+0.5);}

else if(direction == 2){isoccupiedleft = mapHelper.isOccupied(x+0.5,y-1);}

}

if(isoccupiedleft) {

std::cout << "left occupied " << std::endl;

}

//std::cout << "robot.rot= " << tf::getYaw(robotTransform.getRotation()) << std::endl;

//std::cout << "robot.x= " << robotTransform.getOrigin().x() << std::endl;

//std::cout << "robot.y=" << robotTransform.getOrigin().y() << std::endl;

}

Formun Üstü

Formun Altı

**Obstacle Avoidance**

**obstacle\_avoidance.h:** Added new variables.

* bool is\_turning ; Is robot currently turning
* int next\_direction; Target facing direction of the robot

**obstacle\_avoidance.cpp:** Turn 90 degrees right if obstacle is detected

void ObstacleAvoidance::init() {

is\_turning = false;

next\_direction = generalSensorHelper-> direction ;

}

void ObstacleAvoidance::run() {

// 0 +x 0

//3 +y PI/2 1 -y -PI/2

// 2 -x PI

if(is\_turning){

if(generalSensorHelper-> direction != next\_direction){

subsumption = true;

action.turnCommand = -0.4 ;

}else{

subsumption = false;

is\_turning = false ;

action.turnCommand = 0;//stop

}

}else{

subsumption = false;

}

if (generalSensorHelper->isObstacle) {//robot sees an obstacle

// generate obstacle avoidance commands to get rid of the obstacle

subsumption = true;

is\_turning = true ;

next\_direction = (generalSensorHelper-> direction +1)%4 ;

action.forwardCommand = 0;

action.turnCommand = -0.4;//turn right

}

}

Formun Üstü

Formun Altı

**Go to Ball**

**goto\_ball.cpp:** turn to senter and move forward until you see the ball.

void GotoBall::run()

{

subsumption = false;

if (generalSensorHelper->imageHelper.isBallKnown) {

subsumption = true;

int centerx = generalSensorHelper->imageHelper.centerx;

int centery = generalSensorHelper->imageHelper.centery;

// according to balls position on the image generate wheel command to go the ball.

// hint: consider deviation of the ball from the center of the image

action.forwardCommand = 1;

action.turnCommand = 0.4\*(centerx-160)/320; //27

}

}

Formun Üstü

Formun Altı

**Search the Ball**

**search\_ball.h:** Added new variables.

* bool is\_turning ; Is robot currently turning
* int next\_direction; Target facing direction of the robot
* bool left ; If true robot will try to turn left
* int straight; If true robot will try to go straight

**search\_ball.cpp:** Go straight or turn 90 degree left depending of the variables   
Go straight until left is unoccupied(No walls to the left).   
If left is unoccupied go left and then move forward until left is occupied   
Start again

If there is an obstacle ahead obstacle subsumes this behavior and turns right. After turning right left wall is still occupied. So behavior can continue without problem.

void SearchBall::init()

{

subsumption = true;

is\_turning = false;

next\_direction = generalSensorHelper-> direction ;

left = true ;

straight = true ;

}

void SearchBall::run()

{

if (generalSensorHelper->imageHelper.isBallKnown) {

subsumption = false;

}

// 0 +x 0

//3 +y PI/2 1 -y -PI/2

// 2 -x PI

if (!(generalSensorHelper-> isoccupiedleft) && left && !straight) {//left is empty

// generate obstacle avoidance commands to get rid of the obstacle

is\_turning = true ;

left = false ;

straight = true ;

next\_direction = (generalSensorHelper-> direction +3)%4 ;

std::cout << " TURN LEFT\_1 " << next\_direction << std::endl ;

action.forwardCommand = 0;

action.turnCommand = 0.4;//turn left

}else { //go straight

std::cout << " DUMDUZ\_1 " << std::endl ;

action.forwardCommand = 1;

action.turnCommand = 0;//turn left

}

if(straight){

//go straight

action.forwardCommand = 1;

action.turnCommand = 0;//turn left

std::cout << " DUMDUZ\_after\_turn " << std::endl ;

if(generalSensorHelper->isoccupiedleft){

straight = false ;

}

}

if(is\_turning){//first turn then go straight

if(generalSensorHelper-> direction != next\_direction){

action.turnCommand = 0.4 ;

action.forwardCommand = 0;

std::cout << " TURN LEFT\_2 " << std::endl ;

}else{

is\_turning = false ;

left = true ;

action.forwardCommand = 1;

action.turnCommand = 0;//stop

}

}

}

## Deliverables

My three runs took 1.14.30,1.15.85 and 1.11.80   
avarage time is 1.14 min

If we implement user keyboard control behavior , since other behaviors cannot suppress keyboard behavior and keyboard behavior would be always active, other behaviors wont matter. So it would be possible to crash obstacles .

Formun Üstü

Formun Altı