МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Системы реального времени на основе Linux»

Тема: «Гензель и Гретель»

Студенты гр. 3303 _____ Гриднева К.Н.

_____ Смолина С.К.

_____ Ханукашвили В.Д.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2018

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студенты: Гриднева К.Н., Смолина С.К., Ханукашвили В.Д.

Группа 3303

Тема работы: Гензель и Гретель

Исходная постановка задачи:

Найти выход на неизвестной карте. Карту запоминать нельзя.

Исходные данные:

На неизвестной карте необходимо найти выход. Строить и запоминать карту нельзя. За собой остаётся след из хлебных крошек.

Ограничения на исходные данные:

- У вас есть скан местности, по которому можно определить, в какую сторону можно двигаться.
- Один раз в некоторый интервал времени за вами остаётся хлебная крошка, символизирующая, что в этом участке карты вы уже были.
- Карту запоминать нельзя.
- Выход определяется сразу, как только он оказывается в радиусе видимости скана.

Дата выдачи задания: 20.10.2018
Дата сдачи реферата: 26.12.2018

Дата защиты реферата: 26.12.2018

Студенты		Гриднева К.Н.
		Смолина С.К.
		Ханукашвили В.Д.
Преподаватель		Филатов А.Ю.

АННОТАЦИЯ

В настоящей работе описана реализация программы «Гензель и Гретель», в которой робот перемещается по карте. В качестве карты местности используется построенная модель лабиринта. Робот использует одометрию и лазер, который находится на голове робота. Лазер снимает показания о том, на каком расстоянии от него находиться препятствия в виде стен лабиринта и хлебные крошки, которые он оставляет за собой через некоторые промежутки времени. Основываясь на показаниях лазера, робот пытается выйти из лабиринта. Для реализации проекта были использованы такие инструменты как: ROS, Gazebo, Rviz.

SUMMARY

This paper describes the implementation of the program "Hansel and Gretel", in which the robot moves around the map. A maze model is used as a terrain map. The robot uses odometry and laser, which is located on the head of the robot. The laser takes readings about how far from it there are obstacles in the form of labyrinth walls and bread crumbs, which it leaves behind at certain intervals. Based on the laser readings, the robot is trying to get out of the labyrinth. To implement the project, tools such as ROS, Gazebo, Rviz were used.

ВВЕДЕНИЕ

Необходимо реализовать программу, в которой роботу требуется найти выход на неизвестной карте.

Ниже перечислены основные задачи, которые необходимо решить, в ходе реализации программы:

- Спроектировать и реализовать карту местности и крупицы песка;
- Выбрать робота, у которого есть лазерный дальномер;
- Поместить робота на карту и «научить» реалистично перемещаться на ней;
- Добавить роботу следующий функционал: генерация хлебный крошки в произвольный момент времени;
- Добиться того чтобы робот, основываясь на данных одометрии и лазерного дальномера, находил выход из неизвестной ему карты местности.

Ограничения на исходные данные:

- Окружающий мир состоит из прямых линий (комнаты, коридоры);
- Один раз в некоторый интервал времени за роботом остаётся хлебная крошка, символизирующая, что в этом участке карты робот уже был;
- Карту запоминать нельзя;
- Выход определяется сразу, как только он оказывается в радиусе видимости скана.
- У робота есть лазерный дальномер, по которому можно определить, в какую сторону можно двигаться, и данные одометрии.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

ROS (**Robot Operating System**) — **Операционная система для роботов** — это фреймворк для программирования роботов, предоставляющий функциональность для распределённой работы. ROS был первоначально разработан в 2007 году под названием *switchyard* в Лаборатории Искусственного Интеллекта тэнфордского Университета для проекта (STAIR). В 2008 году развитие продолжается в Willow Garage, научно-исследовательском институте/инкубаторе робототехники, совместно с более чем двадцатью сотрудничающими институтами

ROS обеспечивает стандартные службы операционной системы, такие как: аппаратную абстракцию, низкоуровневый контроль устройств, реализацию часто используемых функций, передачу сообщений между процессами, и управление пакетами. ROS основан на архитектуре графов, где обработка данных происходит в узлах, которые могут получать и передавать сообщения между собой. Библиотека ориентирована на Unix-подобные системы (Ubuntu Linux включен в список «поддерживаемых», в то время как другие варианты, такие как Fedora и Mac OS X, считаются «экспериментальными»).

ROS имеет две основные «стороны»: стороны операционной системы *ros*, как описано выше и *ros-pkg*, набор поддерживаемых пользователями пакетов (организованных в наборы, которые называются *стек*), которые реализуют различные функции робототехники: SLAM, планирование, восприятие, моделирование и др.

ROS выпускается в соответствии с условиями BSD-лицензии и с открытым исходным кодом. ROS бесплатен для использования, как в исследовательских, так и в коммерческих целях.

2. ИСПОЛЬЗУЕМЫЕ ИНСТРУМЕНТЫ

2.1. Описание топиков

- **/cmd_vel –** топик используется для того, чтобы робот мог двигаться. В данный топик публикуются сообщения типа Twist, которые состоят из линейной – linear и угловой angular – скоростей;
- **/odom –** подписываясь на данный топик, робот получает сообщения об одометрии, которые предоставляют оценку положения и скорости робота в свободном пространстве;
- **/base_scan –** подписываясь на данный топик, робот получает данные с его скана об дальности расположения каких-либо объектов относительно робота;
- **/set_model_state –** в данный топик публикуются координаты модели, которую планируется помещать на «сцену»;
- **/ spawn_sdf_model** – в данный топик публикуется описание модели, которую планируется добавить на «сцену».

2.2. Описание сообщений

- **ModelState –** с помощью данного сообщения задаются координаты модели, которое публикуется в топик set_model_state;
- **SpawnModel –** данное сообщение публикуется в топик spawn_sdf_model и содержит в себе всю необходимую информацию о модели, которую добавляют на «сцену»;
- **LaserScan –** данное сообщение приходит из топика base_scan и содержит в себе данные полученные от лазерного дальномера;
- **Odometry** – данный тип сообщения содержит информацию об оценки положения и скорости робота, которые он получает из топика /odom;
- **Twist** - данное сообщение отправляется в топик и содержит в себе текущую линейную и угловую скорости;

2.3. Описание пакета tf

tf – это пакет, который позволяет пользователю отслеживать несколько координатных объектов с течением времени. tf поддерживает связь между координатами объектов, позволяет пользователю преобразовывать точки, векторы и т. д. между любыми двумя объектами в любой момент времени.

tf позволяет отвечать на такие вопросы, как:

- Где был объект относительно мировых координат, 5 секунд назад?
- Каковы координаты рассматриваемого объекта относительно другого объекта?
- Каково текущее положение объекта на карте местности?

Используемые классы

tf::Transform – предоставляет функционал для преобразования координат;

tf::TransformBroadcaster – позволяет публиковать преобразованные координаты;

tf::TransformListener – позволяет получать преобразованные координаты;

tf::StampedTransform – предоставляет тип сообщения, который публикуется с преобразованными координатами с использованием класса TransformBroadcaster

tf::Quaternion – данный класс реализует кватернион для выполнения линейных вращений (поворотов) объекта.

3. РАЗРАБОТКА ПРОГРАММЫ

3.1. Проектирование мира

В ходе реализации программы была спроектирована карта местности, которая представлена на рисунке 1.

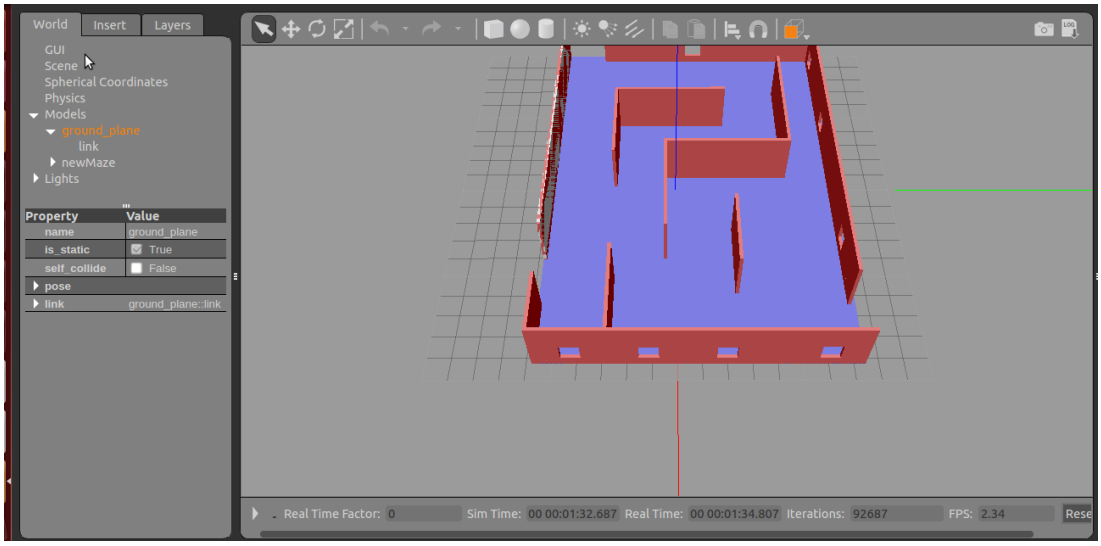

Рисунок 1 – Карта местности

3.2. Реализация основного алгоритма

В начале программы робот помещается внутрь помещения, из которого ему требуется выйти (Рисунок 2).

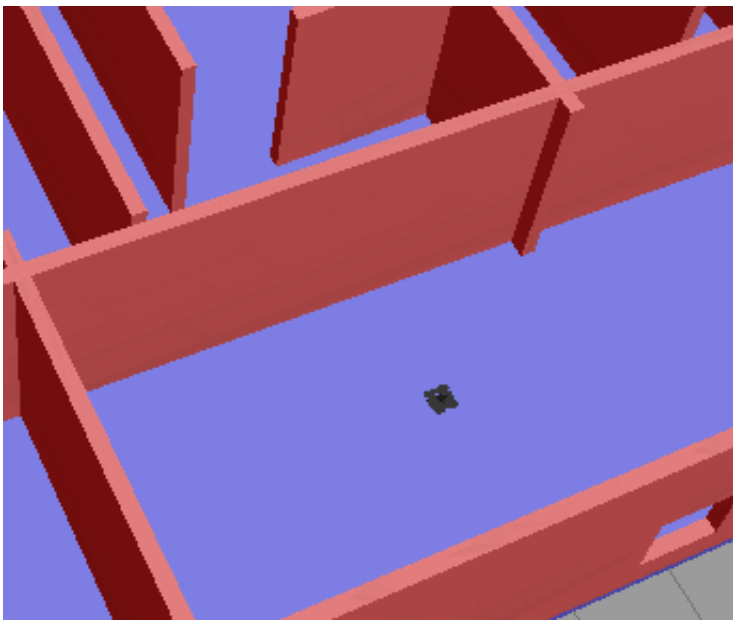

Рисунок 2 – Начало работы программы

Робот начинает перемещаться по карте в непредсказуемом направлении, при этом он просматривает и анализирует расстояние до опасных объектов (стены), оставляет за собой крошки (рисунок 3), смотрит наличие крошек на своем пути – по ним он понимает, что здесь он уже был.

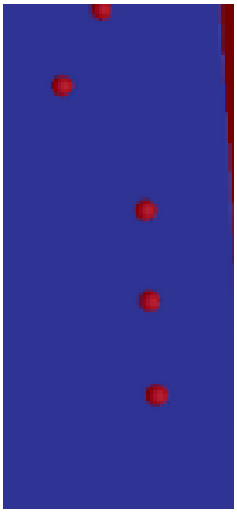

Рисунок 3 – Результат генерации хлебных крошек

В результате работы программы робот находит выход и останавливается перед ним, это представлено на рисунке 4.

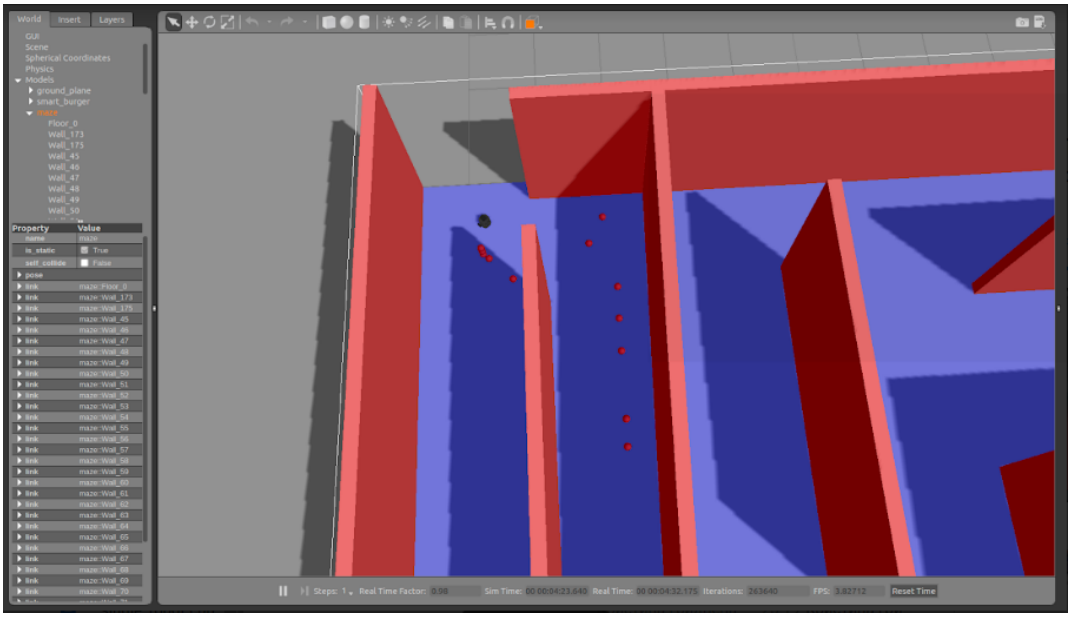

Рисунок 4 – Результат работы программы

3.3. Используемые ноды и топики в rqt_graph

На рисунке 5, с помощью программы rqt_graph, представлены все ноды и топики, используемые в программе.

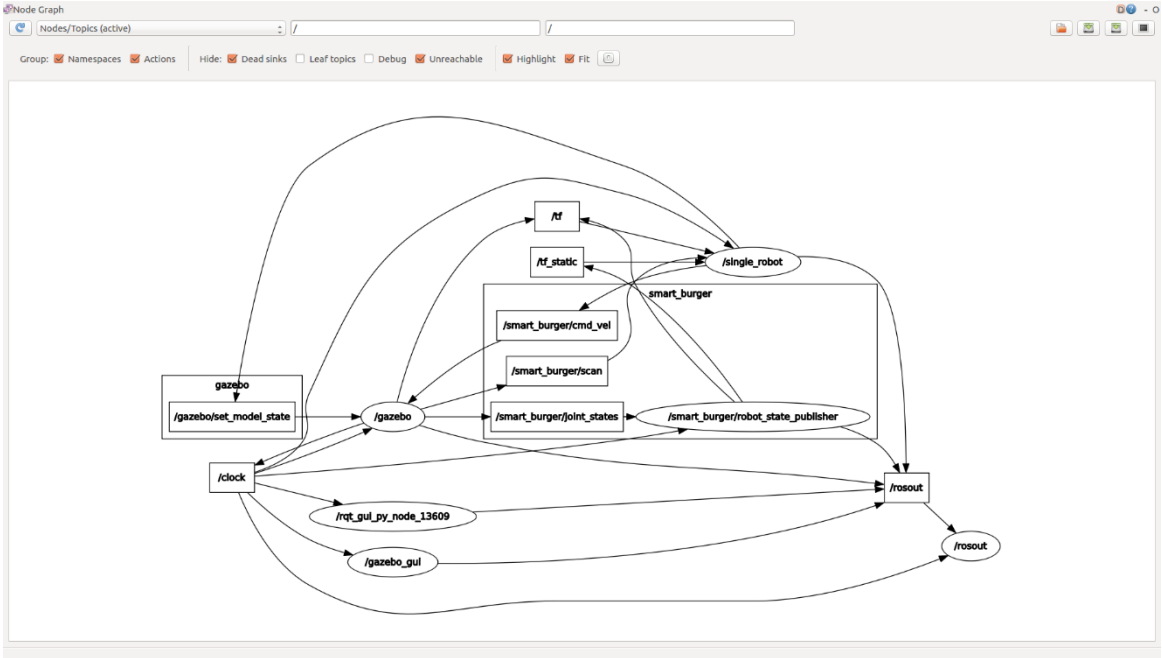

Рисунок 5 – Ноды и топики программы

3.4. Отображение в gazebo

На рисунке 6 представлен снимок результатов работы программы – робот нашел выход.

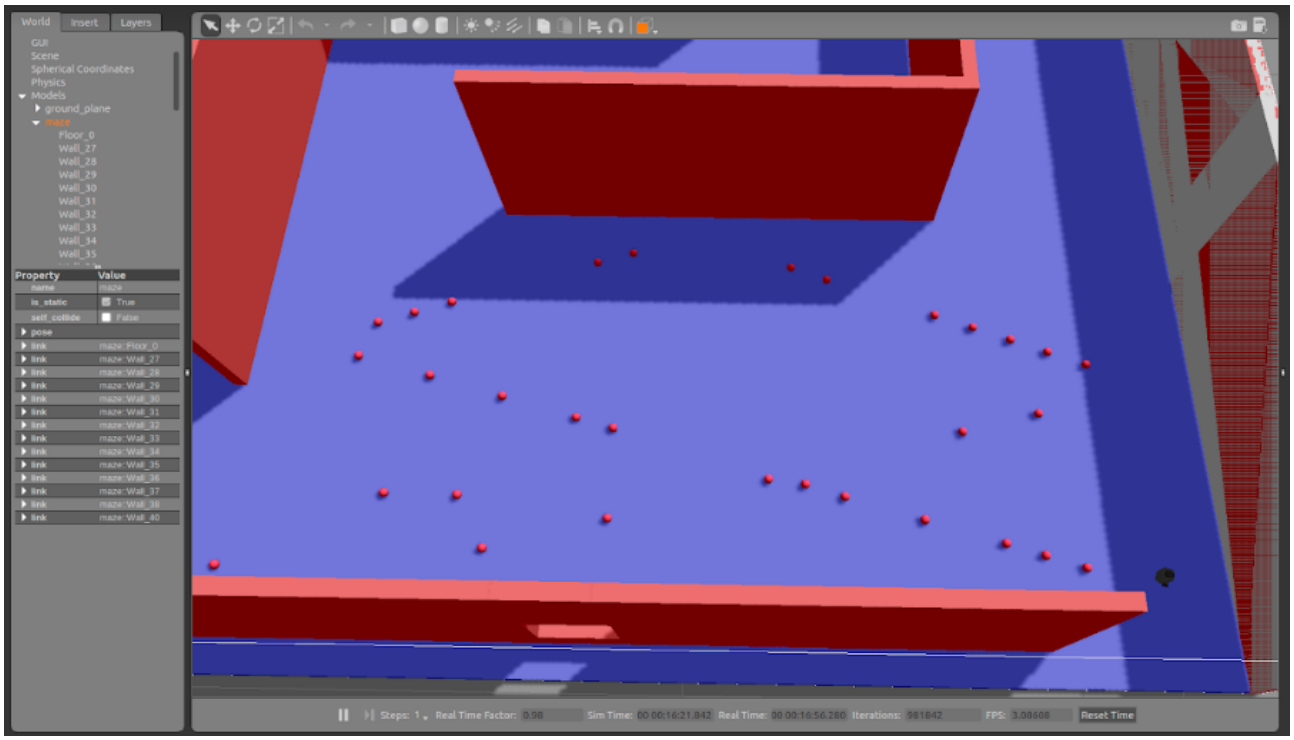

Рисунок 6 – Результат работы алгоритма

ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта была разработана программа, в которой решается задача по поиску выхода из помещения.

Список задач, которые были решены:

- Спроектирована и реализована карту местности и крупицы песка;
- Выбран робот, у которого есть лазерный дальномер - turtlebot3;
- Робот помещен на карту и был реализован алгоритм, который «научил» робота перемещаться по карте местности;
- Роботу был добавлен следующий функционал: генерация хлебный крошки в произвольный момент времени;
- Добились того, чтобы робот, основываясь на данных одометрии и лазерного дальномера, находил выход из неизвестной ему карты местности.

ПРИЛОЖЕНИЕ А

ЛИСТИНГ ПРОГРАММЫ

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="first_tb3"  default="smart_burger"/>

  <arg name="first_tb3_x_pos" default=" 0.0"/>
  <arg name="first_tb3_y_pos" default=" 0.0"/>
  <arg name="first_tb3_z_pos" default=" 0.0"/>
  <arg name="first_tb3_yaw"   default=" 1.57"/>



  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/empty.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

    <!-- Spawn the create by its URDF -->
    <node name="spawn_model" pkg="gazebo_ros" type="spawn_model"
        args="-sdf -file $(find robots)/models/model.sdf -x 0 -y 0 -z 0 -model maze" />

  <group ns = "$(arg first_tb3)">
    <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg
model).urdf.xacro" />

    <node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher" output="screen">
      <param name="publish_frequency" type="double" value="50.0" />
      <param name="tf_prefix" value="$(arg first_tb3)" />
    </node>


    <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-urdf -model $(arg first_tb3) -x $(arg first_tb3_x_pos)
-y $(arg first_tb3_y_pos) -z $(arg first_tb3_z_pos) -Y $(arg first_tb3_yaw) -param robot_description" />
  </group>


</launch>
```

ПРИЛОЖЕНИЕ В

ЛИСТИНГ ПРОГРАММЫ

```
#include "ros/ros.h"
#include "gazebo_msgs/SpawnModel.h"
#include "gazebo_msgs/ModelState.h"
#include <fstream>
#include "string.h"
#include <tf/transform_broadcaster.h>
#include <tf/transform_listener.h>
#include <std_msgs/String.h>
#include <math.h>
#include "sensor_msgs/LaserScan.h"
#include <nav_msgs/Odometry.h>
#include <tf/transform_datatypes.h>
#include <tf/transform_listener.h>

using namespace std;


int sandId  ;
ros::Publisher pubgaz ;
ros::ServiceClient add_model;
gazebo_msgs::ModelState msg;
ros::Publisher cmd_vel_topic;
gazebo_msgs::SpawnModel srv;
//tf::TransformListener listener;


double PI = 3.1415926535897;

struct robot
{
	float p_x;//position
	float p_y;
	float p_z;
	double roll;		//orientation
	double pitch;
	double yaw;
};
struct robot smart;

struct sand
{
	float p_x;//position
	float p_y;
};
struct sand tmpSand;

void setPosition (struct robot *name, float p_x, float p_y, float p_z, double roll, double pitch, double yaw)
{
	name->p_x = p_x;
	name->p_y = p_y;
	name->p_z = p_z;
	name->roll = roll;
	name->pitch = pitch;
	name->yaw = yaw;
	ROS_INFO("robot: %.2f, %.2f, %.2f; rotation: %1.2f, %1.2f, %1.2f", name->p_x, name->p_y, name->p_z, name->roll,
name->pitch, name->yaw);
}

tf::StampedTransform transformPoint2(const tf::TransformListener &listener)  //listener tf smart_burger
{
	tf::StampedTransform transform;
	bool success = false;
	do
	{
```

```
                try{
                listener.lookupTransform("smart_burger/odom", "smart_burger/base_footprint", ros::Time(0), transform);
                tf::Quaternion q(transform.getRotation().x(), transform.getRotation().y(), transform.getRotation().z(),
transform.getRotation().w());
                tf::Matrix3x3 m(q);
                double roll, pitch, yaw;
                m.getRPY(roll, pitch, yaw);
                setPosition(&smart, floor(transform.getOrigin().x()*100)/100, floor(transform.getOrigin().y()*100)/100,
floor(transform.getOrigin().z()*100)/100, roll, pitch, yaw);
                success = true;
                }
                catch(tf::TransformException &ex){
                ROS_ERROR("%s",ex.what());
                ros::Duration(1.0).sleep();
                }
                sleep(1);
        }
        while(!success);
        return transform;
}



void addSand(float posX, float posY)
{
        int  timeAddSand =  rand() % 50;
        ROS_INFO("timeAddSand: %d", timeAddSand);
        if  (timeAddSand%2 == 0)
        {
//Generate uniq name
                stringstream ss;
                ss<< sandId ;
                string st = ss.str();
//Add sand
                srv.request.model_name = st;
        geometry_msgs::Pose pose;
                pose.position.x = posX;
                pose.position.y = posY;
                pose.position.z = 0;
        srv.request.initial_pose = pose;
                add_model.call(srv);
    //Spawning finished

        //poinCoord.push_back(pose);


        msg.model_name = st;
                msg.pose.position.x = posX;
                msg.pose.position.y = posY;
        pubgaz.publish(msg);
                sandId +=1;
                ROS_INFO("sandId: %d", sandId);//cout<<"SandId"<< sandId <<"\n";
        }
}



void move (ros::Publisher cmd_vel_topic)
{
        geometry_msgs::Twist vel_msg;
        float cur_x = round(smart.p_x);
        float cur_y = round(smart.p_y);
        if((cur_x != -10) && (cur_y != 9))
        {
                vel_msg.linear.x = 0.05;

        }
        else {vel_msg.linear.x = 0;}
```

```
		vel_msg.linear.y = 0;
		vel_msg.linear.z = 0;
		vel_msg.angular.x = 0;
	vel_msg.angular.y = 0;
		vel_msg.angular.z = 0;
		cmd_vel_topic.publish(vel_msg);
		ROS_INFO("cur_x: %.2f", cur_x);
		sleep(1);
		tf::TransformListener listener;
		transformPoint2(listener);
		sleep(1);

}

void rotate(float angle, ros::Publisher cmd_vel_topic, bool clockwise)
{
		float speed = 10.0;
		//Converting from angles to radians
		float angular_speed = speed*2*PI/360;
		float relative_angle = angle*2*PI/360;
		geometry_msgs::Twist vel_msg;
		vel_msg.linear.x=0;
	vel_msg.linear.y=0;
	vel_msg.linear.z=0;
	vel_msg.angular.x = 0;
	vel_msg.angular.y = 0;
		if (clockwise) {vel_msg.angular.z = abs(angular_speed)*(-1);}
		else {vel_msg.angular.z = abs(angular_speed);}

		//Setting the current time for distance calculus
		float t0 = ros::Time::now().toSec();
		float current_angle = 0;
		if (relative_angle != 0) {
			while (current_angle < relative_angle)
			{
				cmd_vel_topic.publish(vel_msg);
				float t1 = ros::Time::now().toSec();
				ROS_INFO("My t1: %.2f", t1);
				ROS_INFO("My t0: %.2f", t0);
				current_angle = angular_speed *(t1 - t0);
				ROS_INFO("My current_angle: %.2f", current_angle);
				ROS_INFO("My relative_angle: %.2f", relative_angle);
				sleep(1);
			}
		}
		vel_msg.angular.z = 0;
		cmd_vel_topic.publish(vel_msg);
		tf::StampedTransform transform;
		tf::TransformListener listener;
		transformPoint2(listener);
		//ros::spinOnce();
}


void coordinates_sand()
{
		float cur_yaw = round(smart.yaw);

		if( (cur_yaw >= 0) && (cur_yaw <= 180) )
		{
			tmpSand.p_x = smart.p_x;
			tmpSand.p_y = smart.p_y - 0.5;
		}
		if( (cur_yaw <= -0) && (cur_yaw >= -180) )
		{
			tmpSand.p_x = smart.p_x;
			tmpSand.p_y = smart.p_y + 0.5;
```

```
        }


         ROS_INFO("sand_pose: %.2f, %.2f",tmpSand.p_x, tmpSand.p_y );

}

void scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan)
{
        tf::TransformListener listener;
        bool dangerIsNear = false;
        float sizeRange = scan->ranges.size();

        for (int i =0; i< 90; i++){
                if (scan->ranges[i] < 1.3){
                        //Wall or dif object is near
                        dangerIsNear = true;
                        cout<<"Min size"<< scan->ranges[i] <<"\n";
                }
                else {dangerIsNear = false;}
        }
        for (int i =270; i< 360; i++){
                if (scan->ranges[i] < 1.3){
                        //Wall or dif object is near
                        dangerIsNear = true;
                        cout<<"Min size"<< scan->ranges[i] <<"\n";
                }
                else {dangerIsNear = false;}
        }
        if (!dangerIsNear)
                {
                        transformPoint2(listener);
                        coordinates_sand();
                        addSand(tmpSand.p_x, tmpSand.p_y);
                        sleep(1);
                        move (cmd_vel_topic);
                }
        else {rotate(90.0, cmd_vel_topic, true);}
}


int main(int argc, char** argv) {
        ros::init(argc, argv, "single_robot");
        ros::NodeHandle node;
        ros::Rate loop_rate(10);
        srand (time(NULL));
        int min = 1;
        int max = 5;
        sandId = 1;
    pubgaz = node.advertise<gazebo_msgs::ModelState>("gazebo/set_model_state", 10);
        add_model = node.serviceClient<gazebo_msgs::SpawnModel>("gazebo/spawn_sdf_model");
        cmd_vel_topic = node.advertise<geometry_msgs::Twist>("smart_burger/cmd_vel", 1000);
        //ros::Subscriber sub = node.subscribe("smart_burger/odom", 100, chatterCallback);
        ifstream fin("/home/kris-grid/.gazebo/models/modelsand.sdf");
        //vector<geometry_msgs::Pose> poinCoord;//Vector with sands coord
    string model;
    string buf;
    while(!fin.eof()){
       getline(fin, buf);
       model += buf + "\n";
    }
    srv.request.model_xml = model;
        sleep(1.0);
        ros::Subscriber laserSub = node.subscribe("smart_burger/scan",30,scanCallback);
        ROS_INFO("----------START----------");

        ros::spin();
        return 0;}
```