ROS Programing Report
Autonomous Mapping

Blanchon Marc - Miralles Archibald- BScV

Summary

ROS Programming on TurtleBot

Subject: Autonomous group project, implement a program to Autonomously create a map in unknown environment on a TurtleBot using ROS.

ROS Programming on TurtleBot

Abstract

This Report is an end of a project report. In fact, we can't say that this is the end of the project because the way of study and the final code is not convincing for us.
The report is a state of the art on our project, just reminding and explaining our way of thinking and also our working line. Starting from the learning of ROS really briefly, then explaining our ideas and explaining codes and also the thinking that lead us to develop this code and this way.

We can close this abstract by saying that we took a lot of pleasure working on robots and learning all kind of new processes, and we will hope that in this report it could be extracted this spirit.

Preliminary Work

In this Preliminary Work section, we will explain the beginning of the project, passing from learning the language (briefly described and without code) and our ideas and the feasibility, to the first program with some explanations.

First Approach of ROS

ROS (Robot operating system) is just a way of developing software for a robot. We learnt this way of programming on a robot called TurtleBot, and we will stick to this robot for the whole project.

ROS can be schematized by additional commands, managing packages and built-ins programs on the robot using the command prompt.
We can develop in two different languages as known as Python and C++.
So we can conclude this brief introduction on ROS, by saying that thins way of programming permits a lot of things, such as, adaptability, portability and optimization of programming.
This is kind of simple to understand that ROS is a tool to makes the robot management and programming more easy, it also gives us choice to program in the language we want. Without entering in the deep programing explanations of ROS (because there are specific commands to program in standard languages), we can already guess that, all will be mort simple.
To end, we can also remark that the work is simpler because lot of people worked on libraries and this is free to get help on internet and github, see some built packages, and study the way it works to mimic and understand by practical application.

Project Ideas

In this academic year, we had to find a project to realize with TurtleBot, autonomous project, and we had to create this project during one semester.
First we can add a precision, this project is not reinventing the wheel, lot of library and functionalities are already implemented and it already takes a lot of amount of time to team of researcher to implement these codes. In fast we were only two persons, working on project, so we hadn't resources, time, and knowledge to implement such difficult algorithm.

With these points fixed, we thought about a project, which will not take millions of hours to realize, but such ambitions to be tricky and interesting.
After a lot of talk and compromise, we stick to our project and found his entitle: **Autonomous Mapping in Unknown Environment**.

With this ambition, we had a lot of ideas, lot of possibilities, and lot of way to fail instead of ending the project. But with some reflexions we was sticking to this, we choose our guidelines and objectives.

Objectives:

- The Robot have to be able to move alone, entering one command
- TurtleBot needs to avoid obstacles
- Use the maximum resources we have
- Build the map autonomously
- Follow multiple algorithm to be able to extend the project to bigger dimension

After setting objectives, we had a lot of ideas, some was good, some not. But we will explain this during the whole report.
In fact, all the objectives was clear, but two of them stays blur so we will explain these.

<u>Use the maximum resources we have:</u>

This objective is simple, the TurtleBot is a really powerful robot, he is equipped with lot of accessories, such as Lidar, Kinect, and so on...
We give only these two because this is our concern in this case.
We had the idea to use Lidar, to build the map efficiently, and to separate the obstacles avoiding using the Depth Cloud of the Kinect. This was interesting to separate both applications in a way of optimization, and adaptability.
There is also a Black Cloud on this, we didn't know how to manipulate and extract data from Depth Cloud.

Follow multiple algorithm to be able to extend the project to bigger dimension:

Another blur objective, we had the idea to use algorithm implementation to gives some kind of intelligence to the TurtleBot movements. Obviously as we said, “extend the project to bigger dimension”, this is an ambitious sentence of only two students but it was our way of thinking. In fact, we know that the project would be experimented on controlled and closed little areas. But in our mind, the programing is on another level that only sticking to the asked work.
Despite of thinking easy and saying that it will work on our little testing area, we was thinking about making the robot working on big and really unknown areas, and then restrict the robot to be in the project rules. We can already say that the restriction was easy, because our robot avoids obstacles, and in the testing area, the arena is contoured by walls, so this is obstacles for the robot.
With this rule we followed, we were maybe ambitious, but we was thinking about wider way of programing and we was asking us “What a Professional Programmer would do?”

First program on project

In this section, we will talk on our first program. But we weren’t focusing on project mainly on the first lab, so we were thinking before all, what would optimize the program running.

```
1
2   #!/bin/bash
3   echo "Launching SSH..."
4   #sudo ssh turtlebot@192.168.0.100
5
6
7
8   sshpass -p 'napelturbot'  ssh -T turtlebot@192.168.0.100
9   roslaunch turtlebot_bringup minimal.launch
10
11
```

This is a bash file, this program was about optimizing our time and the launch of the robot. With this program, we were able to open the SSH to TurtleBot, and then launch a command (here minimal launch). This little file, takes time to develop, but it permits us to save time later.

Autonomous Mapping

In this section, we will focus on the work done. But only on the Project. This is a kind of schedule, respecting the order of appearance of our programs.
Obviously we will not show only working programs, to be efficient on a project, we have to fail, and fail, find errors, repair them, and find another problem.
In our way of thinking, failing is the process of succeed.

First Approach: Python

To start the project correctly, we had to find the language we wanted, and to know where to start.
We choose python because we had knowledge on this language (on C++ too but…) and because he seems to be more easy to manipulate the ROS Packaged, we will also see that we didn't stay Python Programmer long, because our guess was wrong.

First we will start by showing includes, and then the main.

Includes:

```
1   #!/usr/bin/env python
2
3   import rospy
4   from geometry_msgs.msg import Twist
5   from sensor_msgs.msg import Image
6   from depth_cam_tools.kinect1 import Kinect1
7   from depth_cam_tools.kinect2 import Kinect2
8   from math import pi
9
10
11  linear_speed = 0.2
12
```

So this is a python file so we just put the first line to be able to run it easily using Command Prompt.
Then we import, ROS Python library, and some packages from Twist, Image, Kinect Classes, math and pi, and define a linear_speed=0.2 which will be the speed of robot, slow but like this robot was controllable.

Main:

```
217 if __name__ == '__main__':
218     try:
219
220         print (type(Image))
221         print Image
222         print (type(Kinect1))
223         print Kinect1
224         print (type(Kinect2))
225         print Kinect2
226
227         key=input("Direction")
228
229         if key == 8:
230             GoForward()
231         if key == 2:
232             GoBackward()
233         if key == 6:
234             GoRight()
235         if key == 4:
236             GoLeft()
237
238     except:
239         rospy.loginfo("GoForward node terminated.")
240
```

We can see that, we try each lines, and if this is not working, then we just transmit error. Then we use input to input something with keyboard, and in function of the input, we do an action corresponding to the wished reaction (if we take a numeric pad, we can see that there is a logic in the choose of number to type and input).

To avoid redundancy, we will only display one function, because this is all time the same principle and this is not necessary to show the whole code if everything is a kind of copy paste changing only two or three values.

```
class GoForward():
    def __init__(self):
        # initiliaze
        global linear_speed
        rospy.init_node('GoForward', anonymous=False)

    # tell user how to stop TurtleBot
    rospy.loginfo("To stop TurtleBot CTRL + C")

        # What function to call when you ctrl + c
        rospy.on_shutdown(self.shutdown)

    # Create a publisher which can "talk" to TurtleBot and tell it to move
        # Tip: You may need to change cmd_vel_mux/input/navi to /cmd_vel if you're not using TurtleBot2
        self.cmd_vel = rospy.Publisher('cmd_vel_mux/input/navi', Twist, queue_size=10)

    #TurtleBot will stop if we don't keep telling it to move.  How often should we tell it to move? 10 HZ
        r = rospy.Rate(10);

        # Twist is a datatype for velocity
        move_cmd = Twist()
    # let's go forward at 0.2 m/s
        move_cmd.linear.x = linear_speed
    # let's turn at 0 radians/s
    move_cmd.angular.z = 0

    # as long as you haven't ctrl + c keeping doing...
        while not rospy.is_shutdown():
        # publish the velocity
            self.cmd_vel.publish(move_cmd)
        # wait for 0.1 seconds (10 HZ) and publish again
            r.sleep()

    def shutdown(self):
        # stop turtlebot
        rospy.loginfo("Stop TurtleBot")
    # a default Twist has linear.x of 0 and angular.z of 0.  So it'll stop TurtleBot
        self.cmd_vel.publish(Twist())
    # sleep just makes sure TurtleBot receives the stop command prior to shutting down the script
        rospy.sleep(1)
```

So we can just look at the main instructions step by step.
We enter a name of node, just a matter of having something clear if we want to seep nodes in Command Prompt through SSH while robot is running.
We declare and also write that, to stop robot we just have to Ctrl+C like always but we have to inform user if he is not friendly with the robot.

We declare the publisher using rospy.Publisher(…), this is necessary to publish in Twist in our case, the commands we want. In fact this is what makes the robot receiving and understanding our instructions.
And then to finish we just set values of speed and angular command to say that he don't have to rotate and to go forward.

This program is really easy to understand, it helps us much to understand the working principle of Publisher, nodes and linear/angular commands.
Just to describe the other moves, the robot for left and right, was rotating respectively with defined speed 1 and -1 respectively and for the go backward, he was just doing 180 degrees on himself and coming back.

But there is a problem in this program, he works, but we enter specific angle, working with specific frequencies of initialization. So this is not generic, and if we remember, this is not in our principle to have something not adaptable.

Another problem is still here, we was using Python arbitrary, and we were thinking that is was easiest. And we were wrong. With this program, with lot of tests and failure, we found that recursive algorithm in python ROS, was really difficult to implement. And obviously, as we said before, we took a lot of help on internet, there were no documentation on python implementation of autonomous, avoiding obstacle robot. So we had to rebuild all alone, and test, and spend hours doing this in python, instead of looking on C++ way of programing where everything is already developed. So we move on and choose C++ development!

There are many advantages about C++ implementation of the code, but, there is also disadvantages.
Python needs nothing to works, only launching in Command Prompt from everywhere on TurtleBot when C++ need many files to run and special ones, special place on TurtleBot.
We gave up Python, for the sake of simplicity, and admit that we had to develop and implement other files to make the robot work.

Second Approach: C++

Before all, in this section, we will display only C++ file, and we will forget about launch files, makelist, XML …
We choose this because we prefer to show only the last final program with all files, instead of overcharge the report with useless files just to show work without point. The real point is to show the program which run, not to fill the place of a report. So in this way of writing the report, we will optimize the pages and the content, and also avoid the unnecessary things displaying.

To add another point, after thinking and many struggling, we changed our mind and forget about the Kinect Depth Cloud, which was really difficult to manipulate and extract, and also to be interpreted. So We will only use Lidar, for mapping, and for obstacle avoiding.

We will start with the main, which is launching other functions.

```
214
215   int main(int argc, char **argv)
216   {
217
218
219       ROS_INFO ("Starting the Autonomous Mapping");
220
221       ros::init(argc, argv, "rplidar_node_client");
222
223       ros::NodeHandle nh_;
224       ros::Subscriber sub;
225       cmd_vel_pub_ = nh_.advertise<geometry_msgs::Twist>("/cmd_vel_mux/input/navi", 1);
226       sub = nh_.subscribe<sensor_msgs::LaserScan>("/scan", 1000, ScanMove);
227
228       ros::spin();
229
230       return 0;
231   }
232
```

In this main we just declare and call functions like always. First we just display the launching on the Command Prompt, then we init node.
We declare the node and the subscriber and we extract messages from Twist, which will be the data to be exploited to makes the robot react as we want.
Then we declare the subscriber and the function (last parameter) to be called and looped.

We finish by ros::spin() to run the program.

Now we will display the big function, obviously this is the first implementation so there is many optimizations that can be done. In a matter of franchise, we let the program as we let him, and we will describe it and the principle (which is not obvious).

We will cut the function in multiple part, so if some instructions aren't closing this is because of this. We will stick to this presentation to be precise and explain each commands and notions separately.

First Part, Declaration, First Scan:

```
void ScanAndMove(const sensor_msgs::LaserScan::ConstPtr& scan)

{
    geometry_msgs::Twist base_cmd;
    base_cmd.linear.x=base_cmd.linear.y=base_cmd.angular.z=0;  //Setting Initial values

    bool status = true; //Boolean to check if the robot have to move
    int available = 1; //Variable to loop
    int count; //Counter
    int savedi=0;
    double saveang;



    while (available < 2) //while loop used to run program without interrupts   //WHILE1
    {

        status = 1;

        count = scan->scan_time / scan->time_increment; //Setting the number of values to be checked in for loop
        //float maxtemp=scan->ranges[0]; //Setting temporary value of the distance between robot and walls

        for(int i = 0; i < count; i++){ //Loop for each points of range //FOR1

            float degree = RAD2DEG(scan->angle_min + scan->angle_increment * i); //Convertion to have the angle for each point in degree

            if ((degree > 150) and (degree < 210)){ //Setting the field of View of the robot //IF1

                if(scan->ranges[i] < 0.30000){ //Looking if obstacle if nearest than 40cm //IF2

                status = 0; //Setting boolean to say that there is and obstacle

                }//ENDIF2

            }//ENDIF1




            //}//ENDIF3


        }//ENDFOR1
```

We can see that the program in commented, but we will explain in this report more accurately. We had to keep in mind that someone can take this program after us without explanations, so comment are mandatory.

Let's start, we input the object scan to our function (this is the Twist Messages), then we declare multiple variables, a message base_cmd, we set (reset) all speed values of the robot at 0, than declare useful variables.

Status is a Boolean which leads to gives us if the robot has to move or not
Available is a variable which is not really useful to break the While loop, we don't know why, but without this loop and breaking condition, the robot is not running.
Count is a counter which is initialized and after will be modified to know the number of ticks the robot will have stocked in Twist Messages.
Savei and Saveang was previous variables that permits us to keep values but this is not used anymore.

Then we can look at the loop.
We are setting the status value to 1 and giving the wished value to count, to increment the for loop just after.
Then we will just describe the functionality of the loop, instead of cutting all in small piece and explaining all. The for loop is used to check each Twist Messages, and extract the angle and translate is from radian to degree, then we have two if. These two if is first to restrict the view of robot, in our case he will see between 150 and 210 (the robot angle is 180 for describing in front of him), and the second if, is to restrict the distance.

So recall all, we loop under the whole Lidar values, we look after only between -30 degrees and +30 degrees of his view, and then, if the range is equal to 0.4 which correspond to 40 cm, then change the status to 0, like this he will pass in obstacle avoiding.

Second Part, status and move management:

Now we will look at a second part (on three parts) of the program. This is the management of movement, and the repercussion of the status change if the loop in first part change status variable value.

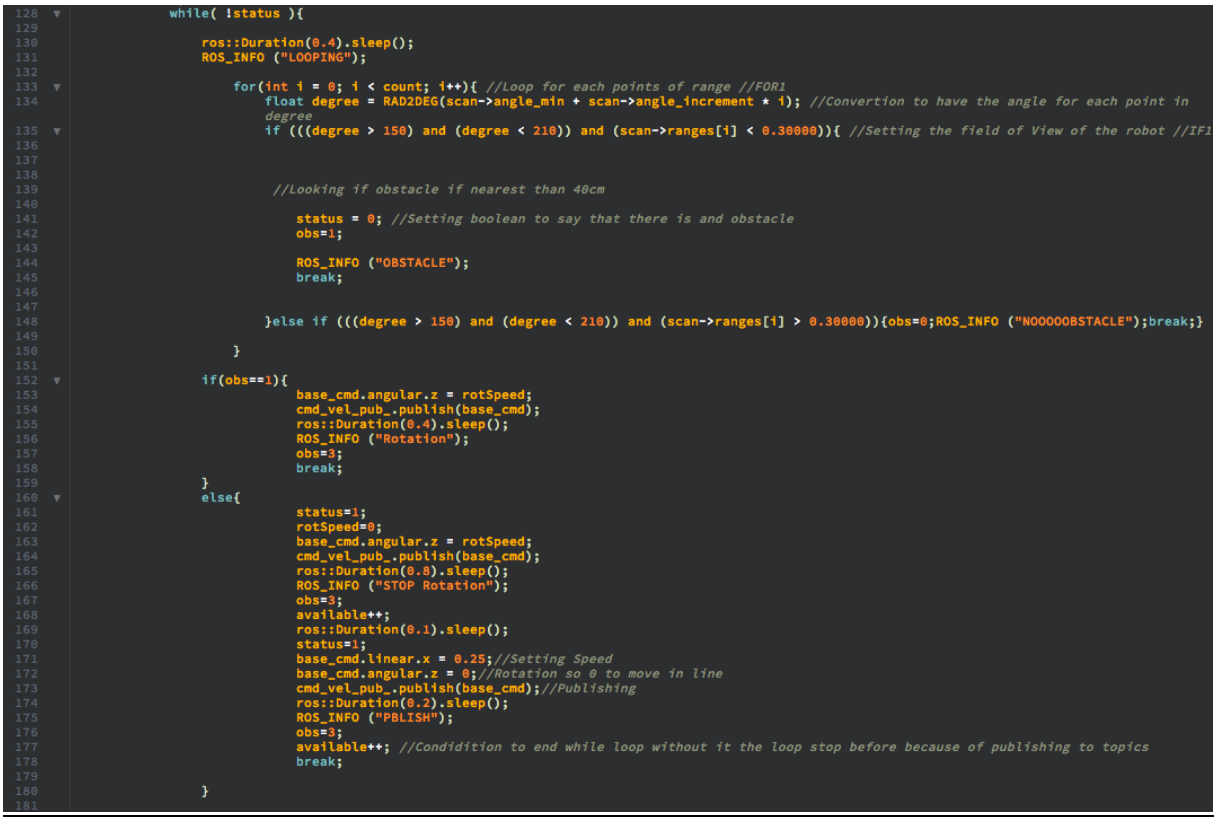
```
103         if (status == 1){//IF4
104
105                 base_cmd.linear.x = 0.25;//Setting Speed
106                 base_cmd.angular.z = 0;//Rotation so 0 to move in line
107                 cmd_vel_pub_.publish(base_cmd);//Publishing
108
109         }//ENDIF4 THEN ELSE1
110         else{//ELSE1
111
112
113
114
115             int randNumber = rand() % 1000 + 0;
116             float rotSpeed=0;
117
118             if(randNumber % 2 == 0){
119
120                 rotSpeed=-0.2;
121
122             }else{
123
124                 rotSpeed=0.2;
125
126             }
127     int obs=3;
```

There we can see, status is the movement manager, if status = 1 then move forward (till no obstacle which is the repercussion of the first part piece of program). So we set linear.x command to 0.25 which is slow speed, we set no angular speed because he have to move in a straight line, and then publish it.

The tricky part start in the else. We had an idea, this is simple to understand but tricky to implement. Let's talk about simple principle, if we want a part of intelligence given to the robot, then we have to include random members.
But to understand why do we have to give this random to robot, we will look at a principle. If we say to the robot to turn only in one direction, there are many special cases where the robot will never end loop between two obstacles, this is a common problem.

So we included random number, when it is even it turns in a way, else et turn other way. This is the principle of this random number and if condition. As we saw before, negative rotation speed will lead to turn right, positive to turn left.
Like this, the robot will never be blocked between two obstacles, obviously he will sometimes look like stuck, but he is not, because randomness always gives results when it is controlled.

Third Part, Movement manager:



```
while( !status ){

    ros::Duration(0.4).sleep();
    ROS_INFO ("LOOPING");

        for(int i = 0; i < count; i++){ //Loop for each points of range //FOR1
            float degree = RAD2DEG(scan->angle_min + scan->angle_increment * i); //Convertion to have the angle for each point in
            degree
            if (((degree > 150) and (degree < 210)) and (scan->ranges[i] < 0.30000)){ //Setting the field of View of the robot //IF1



             //Looking if obstacle if nearest than 40cm

                status = 0; //Setting boolean to say that there is and obstacle
                obs=1;

                ROS_INFO ("OBSTACLE");
                break;


            }else if (((degree > 150) and (degree < 210)) and (scan->ranges[i] > 0.30000)){obs=0;ROS_INFO ("NOOOOOBSTACLE");break;}

        }

    if(obs==1){
                base_cmd.angular.z = rotSpeed;
                cmd_vel_pub_.publish(base_cmd);
                ros::Duration(0.4).sleep();
                ROS_INFO ("Rotation");
                obs=3;
                break;
    }
    else{
                status=1;
                rotSpeed=0;
                base_cmd.angular.z = rotSpeed;
                cmd_vel_pub_.publish(base_cmd);
                ros::Duration(0.8).sleep();
                ROS_INFO ("STOP Rotation");
                obs=3;
                available++;
                ros::Duration(0.1).sleep();
                status=1;
                base_cmd.linear.x = 0.25;//Setting Speed
                base_cmd.angular.z = 0;//Rotation so 0 to move in line
                cmd_vel_pub_.publish(base_cmd);//Publishing
                ros::Duration(0.2).sleep();
                ROS_INFO ("PBLISH");
                obs=3;
                available++; //Condidition to end while loop without it the loop stop before because of publishing to topics
                break;

    }
```

This big loop is the last part of the program. So when status is false, then, he will loop forever. And before all we will explain this. The idea was to makes the robot rotate while he see an object, like this he will never recall all time the big program recurrently, he will just stay in the loop and turn till he didn't see object anymore. This was an idea to optimize program and access. In fact, like this we give a little rotation speed, and if he loop, he will be really precise because he see object, then se turn a little, and so on…

Let's enter in the deep code, we set a sleep to let the robot a little amount of time to think about the previous process (never forget that this is a loop, so we consider the Nth time he passes through this). Then we can see a familiar for loop, in fact this is the same but more optimized so this is the same principle as the second part exactly, the fact is that there is an else condition, which just take account of all the angles and look at distances.
This if else condition is just changing the value which will be a future input.
We can also see breaks in if, this is because we say that, if he sees one time an obstacle, then there is an obstacle, so no needs to scan every points of the Lidar.

After we will use the same process as before, we will not describe every lines, because this was already explained before, but we will explain the concept and we consider the code as the translation of the concept, obviously some errors can be in the code, and this can be a translation error. As we all say as programmer, the main error sit between the computer and the chair. The error is human so we know that the code is not perfect but we tried to have something clean and also something which follow our concept.

So the concept is, if there is an obstacle (obs = 1), then continue to rotate.
Else, if there is no obstacle, then continue one time rotating, then change the rotation speed to 0 and change linear speed from 0 to our wished speed.

As always we publish our commands.

Conclusion on this program:

This program should work but this is theory. There is many many way of optimizing the program, we still think that our concept was good and efficient, but the citation upper explain a lot on this failure.
Instead of stopping there or just continuing on buggy program, we study the messages in command prompt to understand our errors, and after we moved on another thing more optimized and clear.

Before continuing the report, we can also talk about results. The robot was able to move forward, detect obstacle, but when he see something, he never rotate, in fact, when he sees that there was nothing in front of him, he struggle a little, and then continue rotate each time. This looks like smooth when he rotates, but after lot of analyse we could see this little struggle.
The smoothness of this error came from the low speed rotation inputted.

When we study the messages from command prompt of robot, we saw no errors, all went right. So this is why we choose to change our way of thinking, and move to a better program.

Final Program & Explanations

Rebuilding of the Code

As we said before, after failing, let's just rebuilding the base of our program, the obstacle avoiding. Because yes, our main problem is the obstacle.
In this optic, we rebuild from start two new functions, one for obstacle and one for management.

Obstacle:

```
#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"
#include "geometry_msgs/Twist.h"

#define RAD2DEG(x) ((x)*180./M_PI)  //Global defined Function to convert radian to degree

ros::Publisher cmd_vel_pub_;    //Setting Published

bool obstacle(const sensor_msgs::LaserScan::ConstPtr& scan){
    float count=0;
    count = scan->scan_time / scan->time_increment; //Setting the number of values to be checked in for loop

        for(int i = 0; i < count; i++){ //Loop for each points of range //FOR1

            float degree = RAD2DEG(scan->angle_min + scan->angle_increment * i); //Convertion to have the angle for each point in degree

            if ((degree > 150) and (degree < 210)){ //Setting the field of View of the robot //IF1

                if(scan->ranges[i] < 0.30000){
                    return true;
                }
            }
            else if ((degree > 160) and (degree < 200) and scan->ranges[i] > 0.30000) {return false;}
        }
}
```

In this program we can see defines, we just include ROS and the messages.
We define also a function to convert radians to degree, this function was already implemented in the program before, but we prefer to show her because it was obvious in the first program.

We still define our publisher and then we look at our function.
We input the scan to our function, which is the object corresponding to our Twist message.
Then we use exactly the same principle as before, loop for each Lidar position, look at our angle of view and look if you have the good distance.
This is a kind of copy paste, but we knows that is was working. Now we can look at the management.

Management:

```
void ScanMove(const sensor_msgs::LaserScan::ConstPtr& scan){

    geometry_msgs::Twist base_cmd;
    bool bolle=true;

    while(bolle){

        if(obstacle(scan) == false){
            base_cmd.linear.x = 0.25;
            base_cmd.angular.z = 0;
            cmd_vel_pub_.publish(base_cmd);
            ros::Duration(0.1).sleep();
        }
        else{
            while(obstacle(scan) == true){
                base_cmd.linear.x = 0;
                base_cmd.angular.z = 0.4;
                cmd_vel_pub_.publish(base_cmd);
                ros::Duration(0.2).sleep();
                break;
            }
        }


        bolle=false;
    }

}
```

So this function in called in the main, as before and so looped every time. This is all time the same principle we will not change it, this is the way ROS use C++.
So this function is lean, and pretty simple to understand. We had to makes the robot move, so we get rid of the randomness just because we were looking at a working robot, and we will take care of special cases after.

So there is a while, like always, if we remove this, the program is not working and we don't know why. So each time he loops, he will remove condition and stop, then the main will recall function.
And then, this is pretty simple, if he sees something, then rotate, if not, just go forward.
Adding some sleep time to have temporisation of data for the robot and the program is done.
Really simple, really clean, and going to the basis.

Conclusion:

To end this, let's talk about performances. In fact, this program is simple, should work, but it's not the case.
As result, we can observe that he will go in straight line, then see obstacle, rotate correctly, escape from obstacle, but, after a little timer (and we don't know why), he will struggle, like if had to do the both condition at same time.

To be able to understand this, we tried everything, it was obviously not working, so we study the robot answer, and by looking on the robot messages, we saw lot of programs.
Our program never touches or influences the messages, but before struggling all goes well, then without any reason, when he struggles, the messages looks strange, maybe corrupted, some NaN are in the message, some incredible values.
To conclude with this, we tried something lean but we move on and going to our final working program.

Coming back in a previous version

As we said before, let's say that we had to comeback to a version which work to have demonstration to show.
In fact, this version is like all before, this is exactly the same principles, same commands, same way of coding, same inspiration (given program for obstacle avoiding), but this is different because she works.
First let's be honest, this is not our program, all the code lines are mostly from the project of some of our colleagues, they permit us to take the code and change values to have results and demonstration.
We will see that this is exactly the same as us so that's why the code will be displayed but only the changes will be explained.

We are thankful towards Thomas and Anirudht to permit us to have a working project and to exploit this. Obviously this was mandatory to mentioned them and also to say the truth.
We are able to build a program, able to build launches, makefile, everything, because we did it before, but something is definitely not the same because with the same principles and code lines, their program work when our is not.

Before to move to explanations and the code, we would like to mention also another thing.
We had a working personal code, this was great, but he was working on randomness.
So in a sake of honesty, we will explain this and why we didn't exploit this bug and we will move to the Demonstration Code.

Before every try of the codes, also the codes mentioned before in the report, we had a working program, but based on randomness and bugs, we all makes error and this is one.
We was stocking values of our robot where the distance between the robot and an obstacle was the wider. And then we took the angle corresponding to this and input it as angular_z like our program do now.
But all was working well, and a bug in our computer erased this C++ file, and in fact, this was a good thing. Because we studied our program after, and trying to redo the same thing. And After thinking something appear and this is why we are happier to show something working which is not really our, instead of showing something which work, but is totally a bug.

The things that appear was that we were entering a degree value, in the angular-z to move.
So instead of the principle we want: the robot should move to the farest point, the robot was moving randomly in the arena, because he had values as input like 180 instead of having 0.6 for example, and it looks like it was working. But we saw our error and we prefer giving our not working files, and changed files, instead of giving a buggy program, which work randomly.

In fact, we found a solution but we hadn't the time to develop it, it was to include another library and to convert the degree value, into a proportional speed of rotation. And this should work.

After these explanations, we can move to the working files (Demonstration Files).

Explanations: Working Code

C++ file:

```cpp
#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"
#include "geometry_msgs/Twist.h"

#define RAD2DEG(x) ((x)*180./M_PI)                                  //Function to convert the radian reading into degrees.

ros::Publisher cmd_vel_pub_;                                        //Global variable to publish the movement variables.

void ScanAndMove(const sensor_msgs::LaserScan::ConstPtr& scan)
    //Creation of the function for the program
{
    geometry_msgs::Twist base_cmd;                                  //Object to the twist class
        base_cmd.linear.x=base_cmd.linear.y=base_cmd.angular.z=0;
            //Initialize all the movement variables                                     //Variable to store the last
        angle when encountered an obstacle
    float sum = 0;
    float mean = 0;
    bool status = 1;
    //Variable to change between linear and angular movements
    int available = 1;                                          //Variable to check if the program is available
    int count;                                                  //Count for the degrees
    while (available < 2)                                       //Check if the program is available
    {
                count = scan->scan_time / scan->time_increment;
                    //Initialize the count variable
        for(int i = 0; i < count; i++)                          //Start a "for-loop" to check all the data of the
        lidar
        {
                        float degree = RAD2DEG(scan->angle_min + scan->angle_increment * i);
                            //Convert the angle given by the lidar (radian) into degree
                        if ((degree > 110) and (degree < 250))
                            //Check if the angle is in the range [149; 211] which corresponds to one need by the robot to pass between two
                        obstacles at 50cm
            {
                if(scan->ranges[i] < 0.30000)                   //Check if the distance detected is less than 30cm
                {
                                    status = 0;                                 //If it is the case, change the status
                                    variable and store a lastang value corresponding to the current angle
                    sum = sum + (degree-180)*scan->ranges[i];
                }
            }
        }
        ROS_INFO ("Value : %f", sum);
                if (status == 1)                                //Check the status of the robot (1=no obstacles, 0 =
                obstacles met)
        {
            base_cmd.linear.x = 0.25;                           //Set the movement variables for moving forward
            base_cmd.angular.z = 0;
            ROS_INFO ("Move");                                  //Print  a message on the screen to have a visual information
            cmd_vel_pub_.publish(base_cmd);                     //Publish the variables to start moving
        }
        else
        {
                        base_cmd.linear.x = 0;                                  //Stop the robot to moving forward
                        if(sum >= 0)                                        //Check if the last met angle is on the a left
                        part or right part of the robot
```

```cpp
            {
                            base_cmd.angular.z = 0.4;                               //Set the variables for right rotation if
                            lastang is on the left side
                ROS_INFO ("Left Rotation");                         //Print  a message on the screen to have a visual
                information
            }
            else
            {
                            base_cmd.angular.z = -0.4;                              //Set the variables for left rotation if
                            lastang is on the right side
                            ROS_INFO ("Right Rotation");                                //Print  a message on the screen to
                            have a visual information
            }
            cmd_vel_pub_.publish(base_cmd);                         //Publish the twist commands
                        ros::Duration(0.2).sleep();                             //Add a delay to the rotation to make it
                        possible
        }
                base_cmd.angular.z = 0;                                 //Stop the robot rotation
        cmd_vel_pub_.publish(base_cmd);                             //Publish the stop command
                available++;                                        //Set available to false to do the loop only once
    }
}


int main(int argc, char **argv)                                     //Declare the main function
{
    ros::init(argc, argv, "rplidar_node_client");
        //Initialize the lidar node
        ros::NodeHandle nh_;                                        //Initialize the NodeHandle
        ros::Subscriber sub;                                        //Initialize the Subscriber
        cmd_vel_pub_ = nh_.advertise<geometry_msgs::Twist>("/cmd_vel_mux/input/navi", 1);
            //Initialize the movement of the robot
        sub = nh_.subscribe<sensor_msgs::LaserScan>("/scan", 1000, ScanAndMove);
            //Run the program
    ros::spin();

        return 0;
}
```

So here we can see the Demonstration Code, as we said before, we will not explain everything because this is already explained two times in the report, and also because there is only just one small difference and this is the management of direction.
In fact, this is the same principle, if he sees obstacles, he turn, else he go forward, exactly the same.
But the manage right or left moves, this code use a sum which is declared in the detection loop “ sum = sum + (degree-180)*scan->ranges[i]; ”.
This line is only to know if the obstacle is to the right or the left.

This is a trick used, because if he sees something to the right, then the angle will be superior than 180, so the robot will turn left.
This is the invert for turn right this is just a masked trick to know if the obstacle is to the right or the left.

CMakeList:

This file is mandatory when we use C++ file, because it permit to build and some kind render the C++ executable.

```
cmake_minimum_required(VERSION 2.8.3)
project(autonomous_map)

find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
)

include_directories(
  ${catkin_INCLUDE_DIRS}
)

catkin_package()

add_executable(NodeSubAM src/autonomous_map.cpp)
target_link_libraries(NodeSubAM ${catkin_LIBRARIES})

```

So the file is simple, make the project and name it as we want, find the components needed to run the program, include the directories of our project, and then build as package, and makes the file executable (because it is needed by ROS).

Package.xml:

```xml
<?xml version="1.0"?>
<package>
  <name>autonomous_map</name>
  <version>0.0.0</version>
  <description>The autonomous_map package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="turtlebot@todo.todo">turtlebot</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>


  <!-- Url tags are optional, but mutiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/autonomous_movement_bscv_2016</url> -->


  <!-- Author tags are optional, mutiple are allowed, one per tag -->
  <!-- Authors do not have to be maintianers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->


  <!-- The *_depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use build_depend for packages you need at compile time: -->
  <!--   <build_depend>message_generation</build_depend> -->
  <!-- Use buildtool_depend for build tool packages: -->
  <!--   <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use run_depend for packages you need at runtime: -->
  <!--   <run_depend>message_runtime</run_depend> -->
  <!-- Use test_depend for packages you need only for testing: -->
  <!--   <test_depend>gtest</test_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>std_msgs</run_depend>


  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

The only things to remember about this file is, this is mainly definition, name of project, licence, and after definition of the dependencies at the bottom.

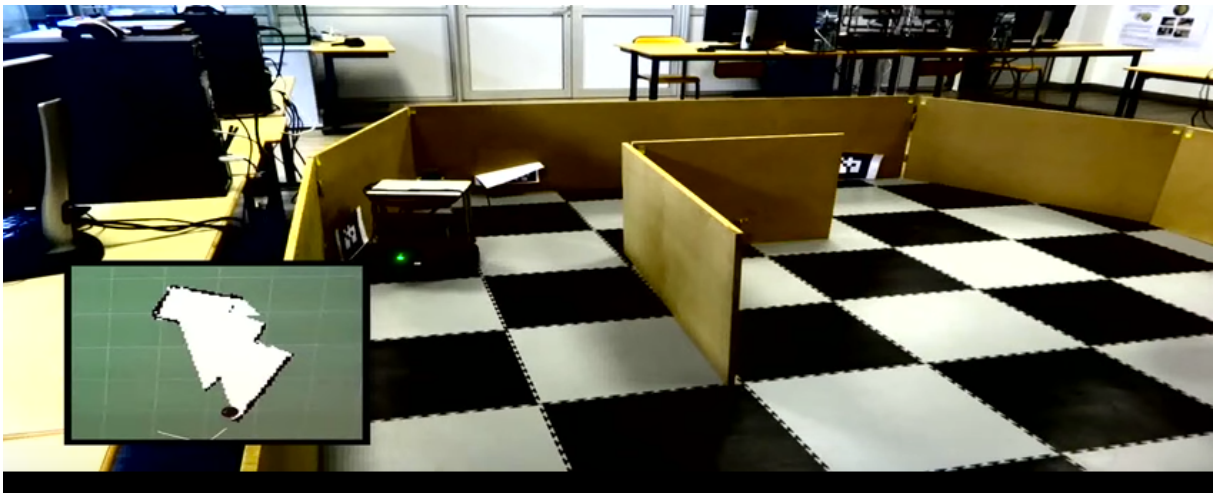
Final File, Launch File:

```xml
<launch>
  <include file="$(find rplidar_ros)/launch/rplidar.launch" />
  <include file="$(find turtlebot_bringup)/launch/minimal.launch"/>
  <node pkg="autonomous_map" name="NodeSubAM" type="NodeSubAM" output="screen">
  </node>

<include file="$(find turtlebot_bringup)/launch/3dsensor.launch">
    <arg name="rgb_processing" value="false" />
    <arg name="depth_registration" value="false" />
    <arg name="depth_processing" value="false" />

    <!-- We must specify an absolute topic name because if not it will be prefixed by "$(arg camera)".
         Probably is a bug in the nodelet manager: https://github.com/ros/nodelet_core/issues/7 -->
    <arg name="scan_topic" value="/scan" />
  </include>

  <include file="$(find turtlebot_navigation)/launch/includes/gmapping.launch.xml"/>

  <include file="$(find turtlebot_navigation)/launch/includes/move_base.launch.xml"/>


</launch>

```

In this file there is many interesting things, the three first lines correspond to the project himself, se he declares the launching of Lidar, launching of minimal launch of the robot, which is mandatory to control him, then launch our C++ file.

After this, all the lines are something which is missing in the report: The Mapping.
Yes, because everything is well done by developers, the mapping can only be performed by adding files in the launch file, and then we open a window to visualize the robot and the map, and the robot will build the map alone without any lines of codes.
After this we can manually save the map on the computer. This is really easy to handle.
We could have developed this kind of process but it would have take thousands of hours.

To end this properly, and see a map building, because images are betted than words we can show the mapping.



And this is the link to see the full video of working robot and mapping.

https://www.youtube.com/watch?v=ClkKL6RNW5o&feature=youtu.be

To end with this section, the results are displayed and accessible through youtube.
Our project went well, the code is working and the robot also, the algorithm is still not perfect, sometimes the robot get caught in infite loop and can't move anymore or takes to much time to be considered as a proper working.

Conclusion

In conclusion, we can just recall everything in a brief way, but we will prefer talking about our personal feelings.
So we created a project more or less from beginning to the end, autonomously, using a new way of programming. And definitely, this works, because we learnt.
Obviously we don't have results all time, obviously we fail. This is how the human learn. And this is pretty ironic to say that a human learns a lot during this project, while he was learning to do something to a robot.

This project learnt us how to manage project, how to fix objectives, improves our programming skills, and this was really interesting.
We realise also that this is a chance to work with robot (which are pretty costly too) and to be free to access to them and work with them, experiment things without real rules except the obvious ones.