M1 MOSIG Robotics

Localization of robair

Team:

CHAKHVADZE Tsotne
TEP Pungponhavoan
CHATTERJEE Jit
DANYSHBEK Assylbek
SHARMA Paritosh

During two week, we had to complete two new nodes: **compute_score_node** and **localization_node**.

In the compute_score_node we had to implement the compute_score function, which computes score depending position and rotation angle.

For each hit of the laser we compute its position in the cartesian space of the framework on the map. And for each hit we check if the computed position corresponds to the occupied cell, if yes we increment score.

//compute position

```
hit.x = x + range[loop] * cos(o + beam_angle);
hit.y = y + range[loop] * sin(o + beam_angle);
```

//check corresponding cell

```
if(cell_value(loop_x, loop_y) == 100)
    cell_occupied=true;
```

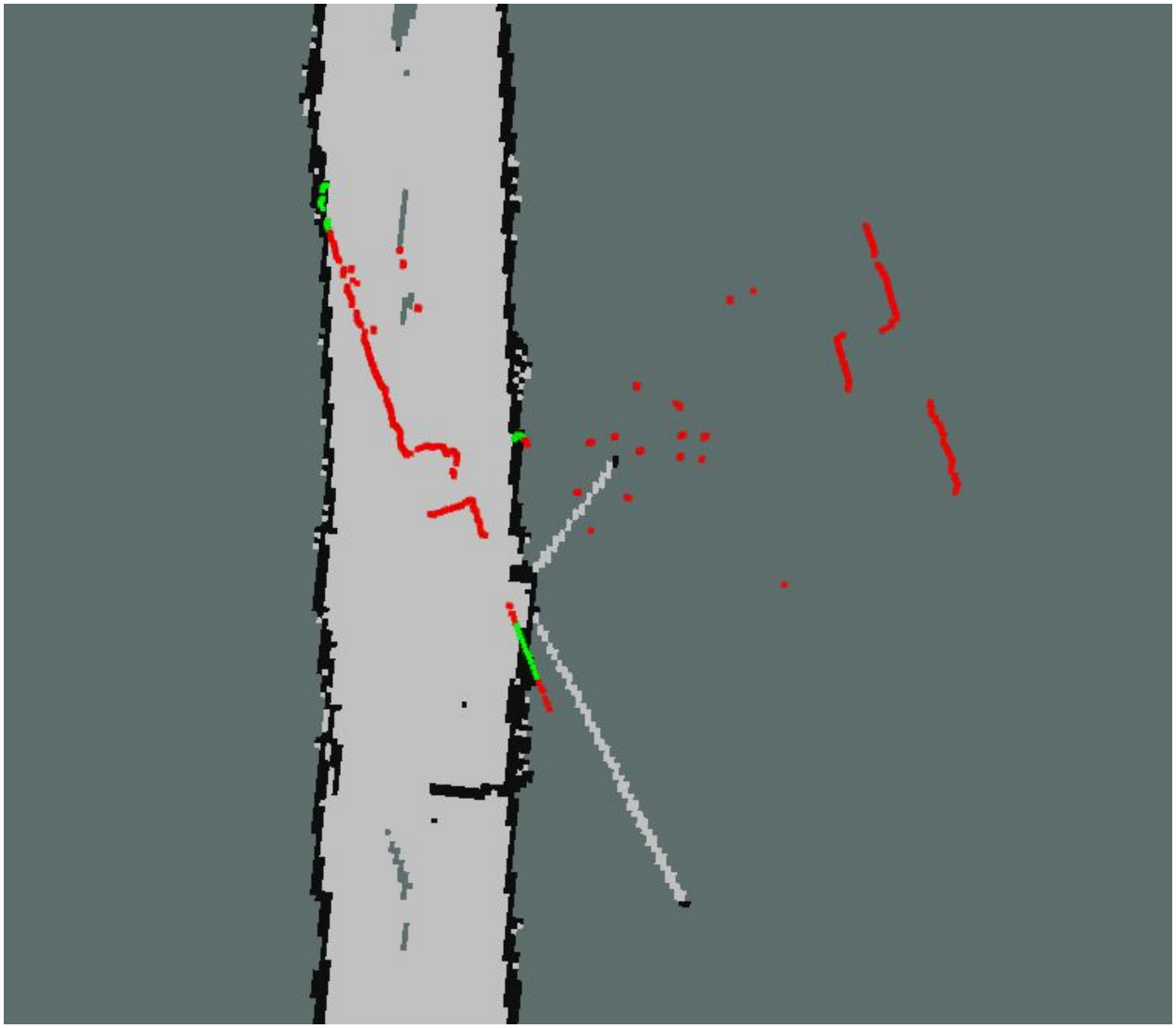
We also edited cmakelists and added that nodes into follow me project:

```
add_executable(compute_score_node src/compute_score_node.cpp)
add_executable(localization_node src/localization_node.cpp)

target_link_libraries(compute_score_node ${catkin_LIBRARIES})
target_link_libraries(localization_node ${catkin_LIBRARIES})
```

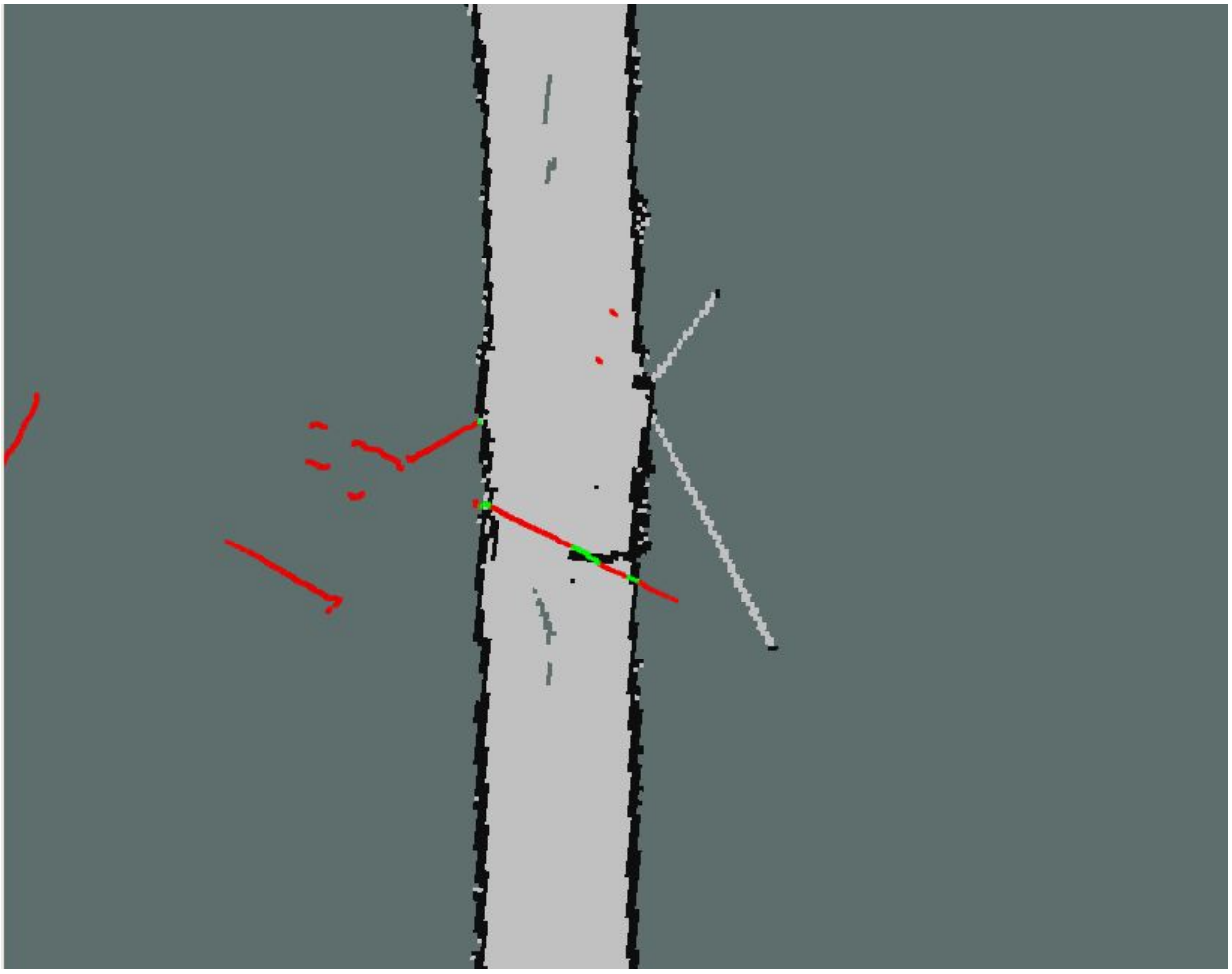
Compute_score_node allows to enter desired point position and angle degree in radians and compute corresponding score, also with help of rviz we visualise it:

```
x (in meter) = 11.45
y (in meter) = 4.4
o (in radian) = 3.1
[ INFO] [1586357923.185901517]: score for position(11.450000, 4.400000, 3.100000) = 48
x (in meter) =
```

Some visualization examples:
For the point above:



For (7,7,19)



Compute_score function is also a key part of **localization_node.** Here, first we make global localisation (find the initial position) and then if the robot moves 1 meter or rotate at least 20 degree, we relocate the position.

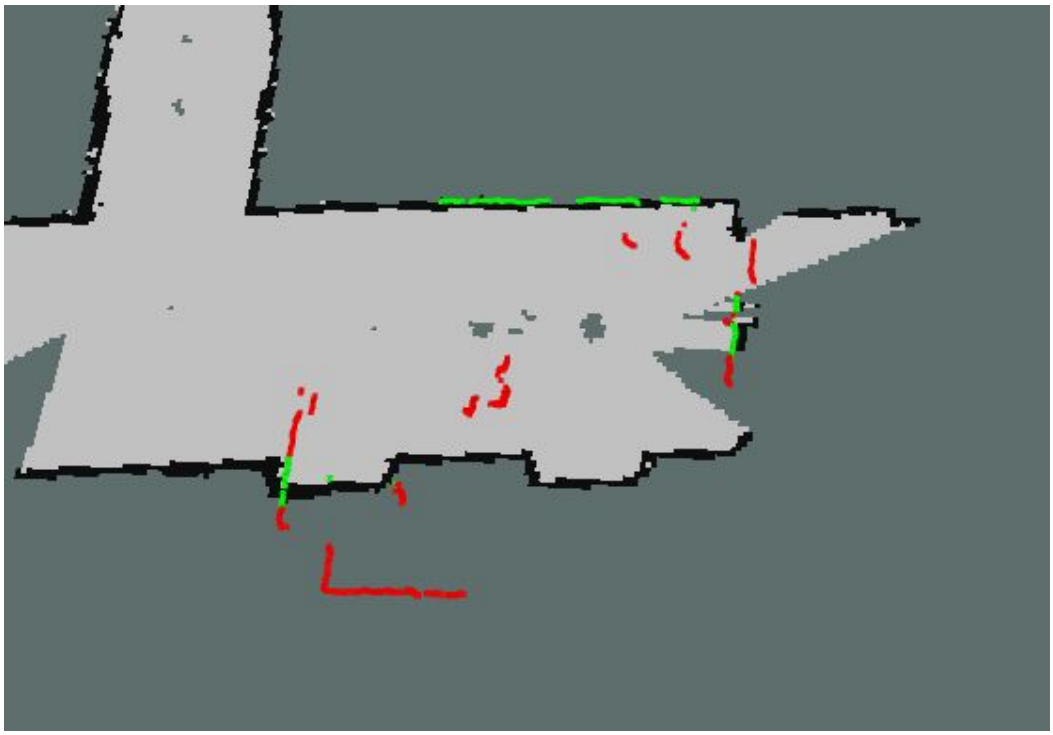
To find the initial position, we check each possible position of the map and choose one, with the highest compute_score.

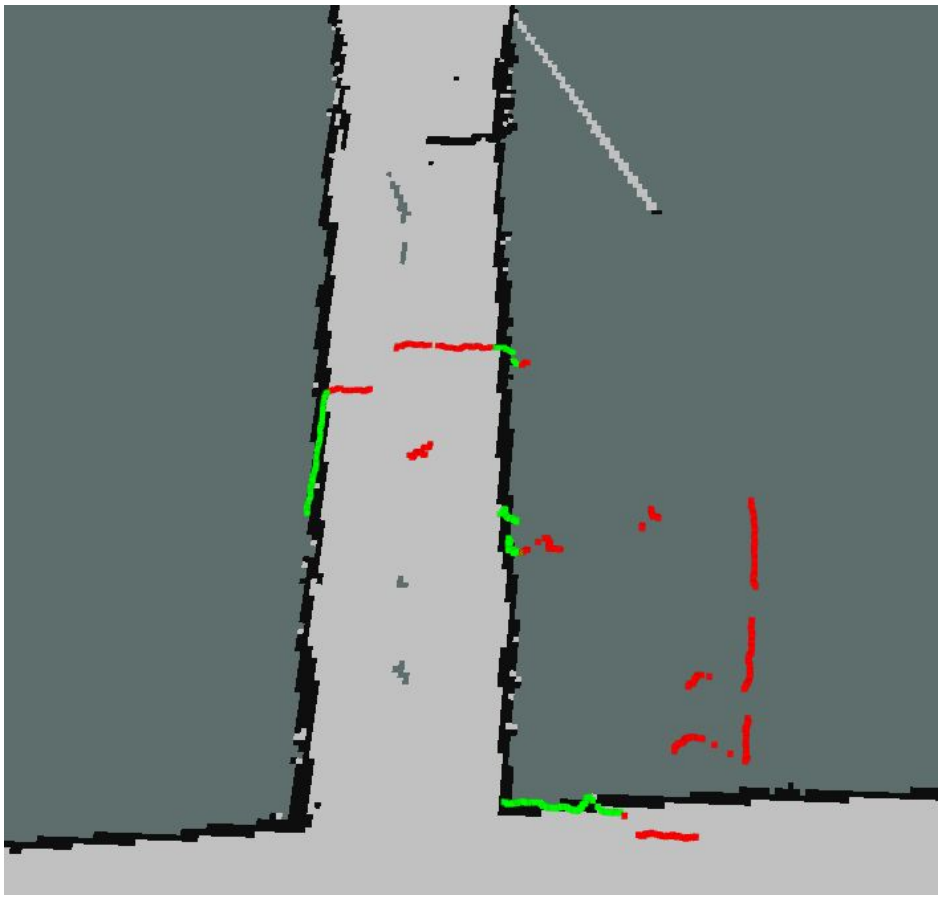
```
for(x=min.x;x<=max.x;x+=0.05)
      for(y=min.y;y<=max.y;y+=0.05)
            {
            if(cell_value(x,y)==0)
            {
                  for(o=0;o<=360;o+=5)
                  {
                        float theta=o*M_PI/180;
```

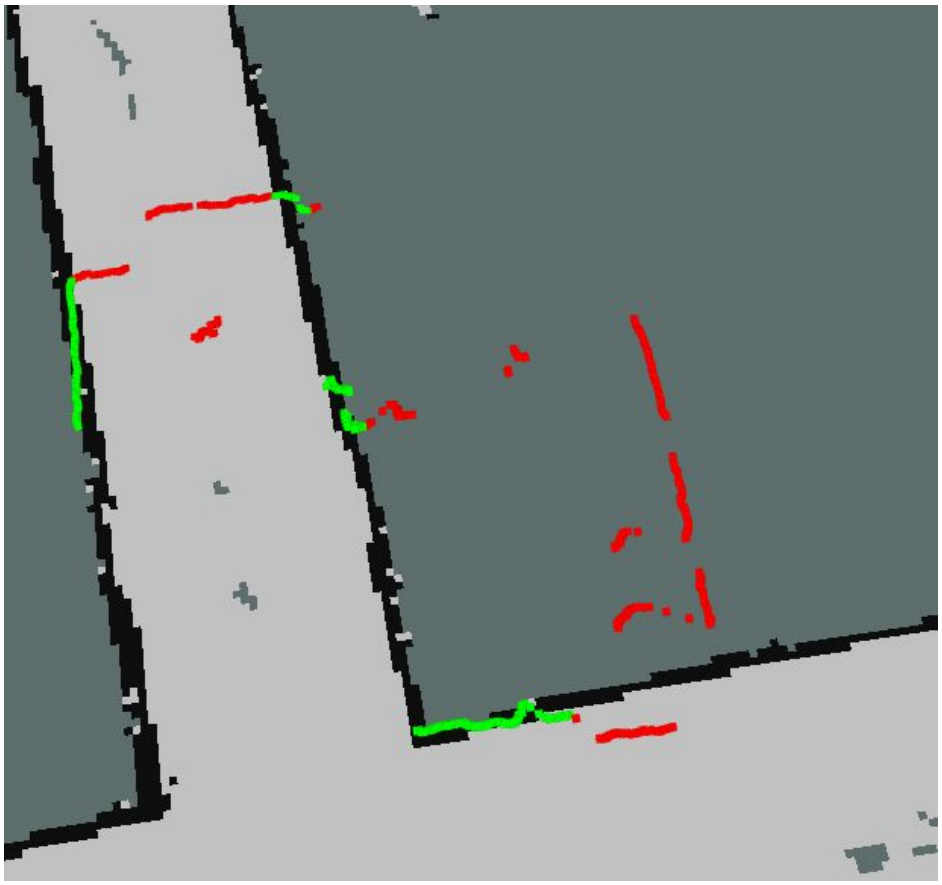
```
                        int curr_score=compute_score(x,y,theta);
                        if(curr_score>score_max)
                        {
                              score_max=curr_score;
                              map.x=x;
                              map.y=y;

                              map_orientation=theta;
                        }
                  }
            }
            }
}
```

(we didn't find how to accelerate process, because 0.05 meter increment take more than 5 minutes, to check how relocation worked sometimes used much higher increment)

Some visualisation what we obtain during initial localization:





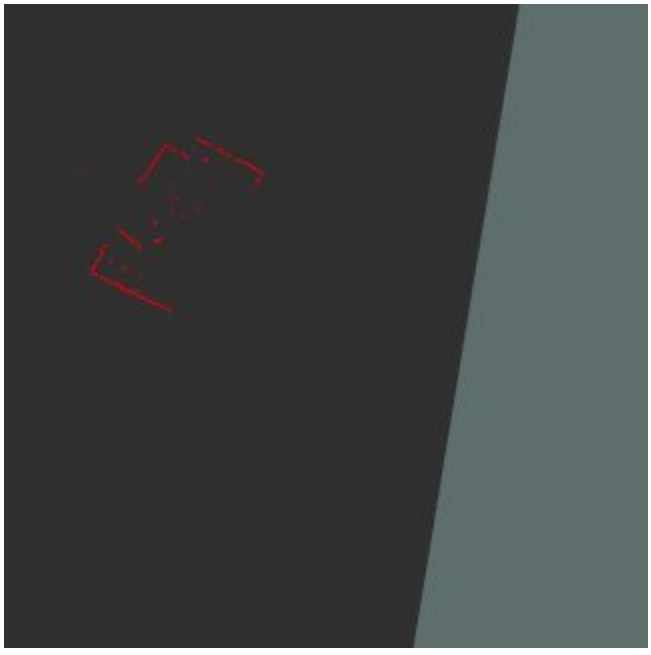


For the relocalize (when the robot moved 1 meter or rotated 20 degree), first we predict that position and then try to find the position around 0.5 meter radius of our predicted position that has the highest compute_score.

```
map.x += distance_traveled*cos(map_orientation);
map.y += distance_traveled*sin(map_orientation);

//to be sure that our predicted position is in the map [width,height] range
map.x=std::min(max.x,map.x);
map.x=std::max(min.x,map.x);

map.y=std::min(max.y,map.y);
map.y=std::max(min.y,map.y);

map_orientation += (odom_current_orientation-odom_last_orientation);

if ( map_orientation < -M_PI )
      map_orientation += 2*M_PI;
if ( map_orientation > M_PI )
      map_orientation -= 2*M_PI
```

Because we do rough prediction, sometimes predicted values are out of range of the map, so we normalize it.

If not we got result:



Because inaccurate and rough prediction, relocalize does not do its job very well, so during the next week we try to implement some of the SLAM algorithm to deal with localization problem.

With the report, we send our code too.