MICRO-453: ROBOTICS PRACTICALS

TP9 Tangible Human-Swarm Interaction using ROS

June 4, 2019

CUI Mingbo

LI Weipeng

ZHOU Xiao

Contents

1 Introduction

The Cellulo robot is an educational robot developed by CHILI Labs. It has a hexagonal symmetrical shape. They are a kind of swarm robots that can be used to simulate the orbit and molecular motion of a planet in the classroom. In this experiment we have to control these cellulo robots so that they can: follow, aggregate, spread.

2 Leader Follower

2.1 Introduction

In this part we implement a simple interactive leader follower behavior. We determine a leader by long pressing the six sensors on top of cellulo, after which the six lights on top of the leader will all turn red. Then another cellulo robot on the map will automatically become the follower, then it will light up six green lights. After that we control the follower so that it always follows the leader's movement pattern.

2.2 Experimental Result

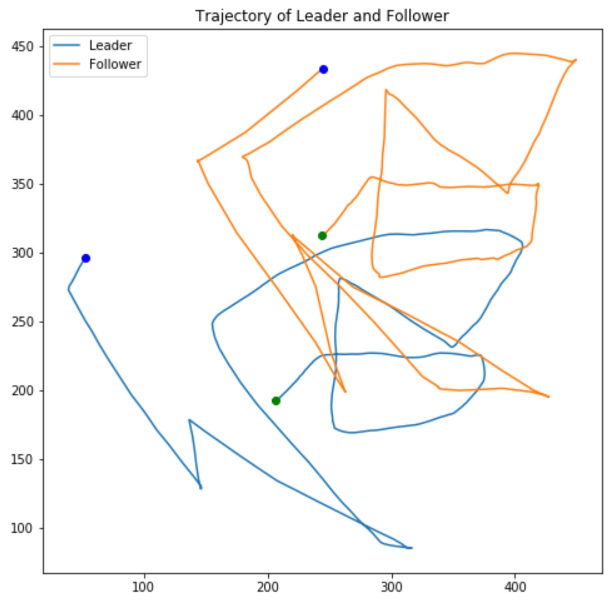

Figure 1: trajectories of the leader and follower

We plot the trajectories of the leader and follower. From Figure 1 the blue one is leader and the orange one is the follower. We can see that the followers follow the leader well. But we can also see that the follower 's trajectory look smoother than the leader's trajectory. This may be because there is a slight delay between the two robots. So when the leader to make a quick turn or jitter, follower will use a more direct route.

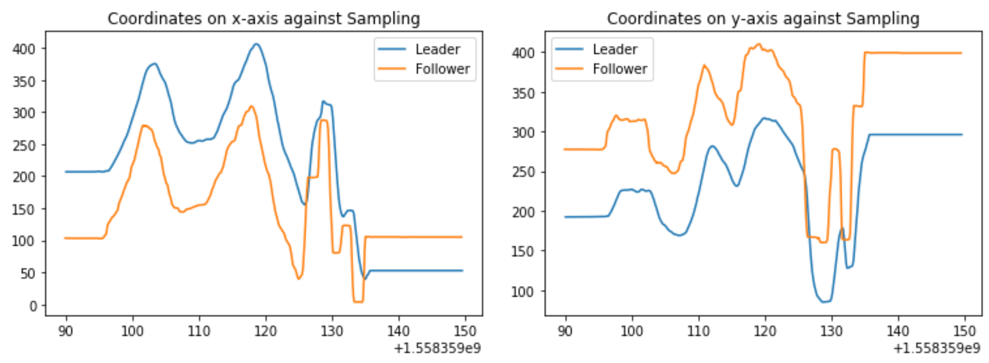

Figure 2: Coordinates of the Leader and Follower on 2 axis against Sampling

Also from the change of coordinates on x-axis and y-axis, it can be observed that the follower can follow the leader with an acceptable performance, despite some little delay.

2.3 Conclusion

Under ideal conditions, the following two conditions should be met between the leader and the follower:

1. The shape of their path of action should be "congruent."

2. And the distance between them should be constant and equal to the initial distance. i.e. $d(t) = d(0) = cte$

From Figure1 we can see that the follower follows well when the leader moves smoothly, but the follower does not follow the leader when the leader moves quickly. This is because of the time delay. The communication between the two robots is discrete, assuming that the time interval between them is: $\tau = t_{n+1} - t_n$. Then the information on how the leader moves during the period of τ will not be known by the followers. If the leader uses a complex and fast method from p^l_n to p^l_{n+1}, the follower will simply and quickly from p^f_n to p^f_{n+1} point. This causes their trajectories to be different. From Figure2 we can see that the distance between the leader and followers is still good, and the bad part may be caused by time delay.

In short, reducing the time interval between communication between robots can fundamentally improve the performance of interaction, or reduce their speed can also effectively improve this.

3 Aggregation

3.1 Introduction

Aggregation is commonly observed in natural swarms, here we implement a kind of aggregation. In swarm robots aggregation, each of the robot positions itself close to each other in one spot. This is done by reducing the distance between them. We cinsider that the Cellulo robotics have two status wait and random walk. At the beginning all the robots were randomly moved until it happened to have a robot close to him. We will adjust different parameters to find a better aggregation method.

3.2 Experimental Result

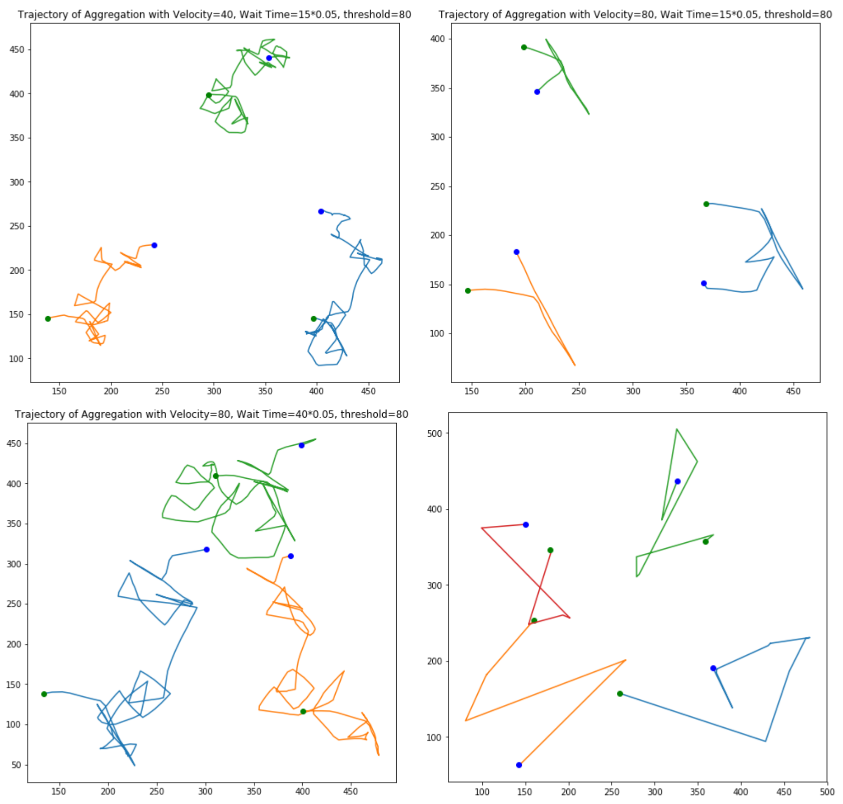

Figure 3: Trajectories of Aggregation with Different Parameters

We plot the trajectories of the 3 or 4 robots by aggregation. From Figure 3 we can observe that the robots has a trend to aggregate together although there exist some local fluctuation for each robot. Also, different parameters can lead to different aggregation performance, in our practical, the third experiment has the best performance among 3-robot aggregation.

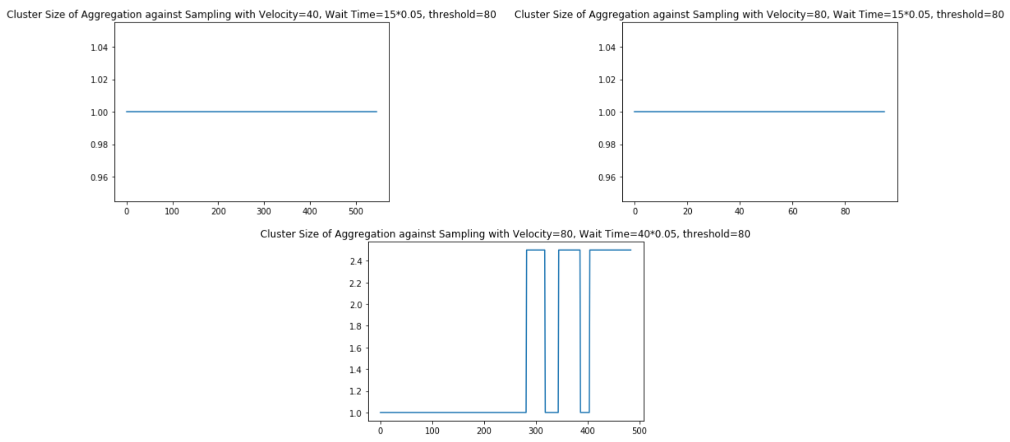

Figure 4: Cluster Metric of Aggregation with Different Parameters with Distance Threshold of 100

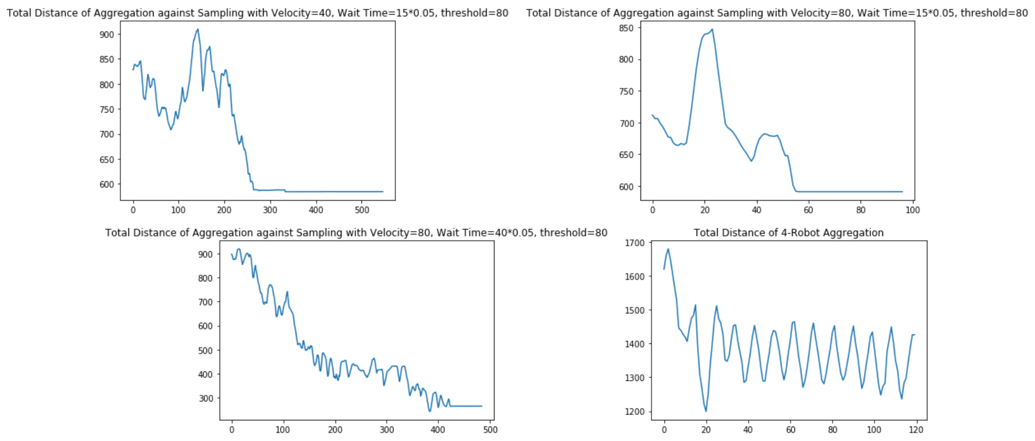

Figure 5: Total Distances of Aggregation with Different Parameters

We can evaluate the aggregation performance by total distances among the robots or the cluster metric. Here we display the most representative ones among our practical. The reduced distance metric and higher cluster value can be represented as a good signal of aggregation. After comparing different sets, we get the influence of different parameters (mainly we tuned the velocity, wait time and threshold) in the code. We assume that the set threshold can determine the final distance among the robots. High speed can lead to more strong local fluctuations when each robot is trying to move to detect the nearby robots, and thus lead to less aggregation time if other parameters are tuned properly. Wait time can be represented as the influence of robot moving in one direction in a period of time. Comparing the first 3 experiments, a proper combination of the 3 parameters can get to an acceptable performance.Also, we try another experiment of same parameters while adding the robot number to 4. We find that the performance is also acceptable despite some local fluctuations. The aggregation of 4 robots can be easier in some cases with properly tuned parameters because each robot can have greater chance to detect the nearby objects before moving for aggregation. Also, the robot is more likely to stay stable because more robots can be located at two different sides of that robot.

3.3 Discussion and Conclusion

In this section, we implement our method to realize the aggregation of multi-robots. We study the influence of different parameters in aggregation, and try to tune the proper parameters to realize good aggregation performance. However, in this method, the trajectory and metric plots show that the robots are not moving in a smooth way and there usually exist obvious local fluctuations for each robot to seek to detect the nearby object, an it may induce some problems like collision in real applications. Thus, a more smooth aggregation with better control algorithm is desired , and in the next part, we will try another aggregation method that is based on attractive potential field.

4 Deployment and Area Coverage

4.1 Introduction

In this part we implement the swarm behaviorâĂŤâĂŤthe Swarm deployment, or dispersion. In this section we use a concept similar to the "field" in physics to manipulate Cellulo robots. We cover one or more âĂIJfieldsâĂİ for the entire area. The closer to the center of the âĂIJfieldâĂİ, the greater the intensity of the âĂIJfieldâĂİ and the greater the âĂIJforceâĂİ the robot receives. The total force that the robot is subjected to is the sum of the effects of these "fields" on the robot. We can use the virtual field to find the virtual force to know and control the acceleration, speed and position of the robot.

4.2 Experimental Result

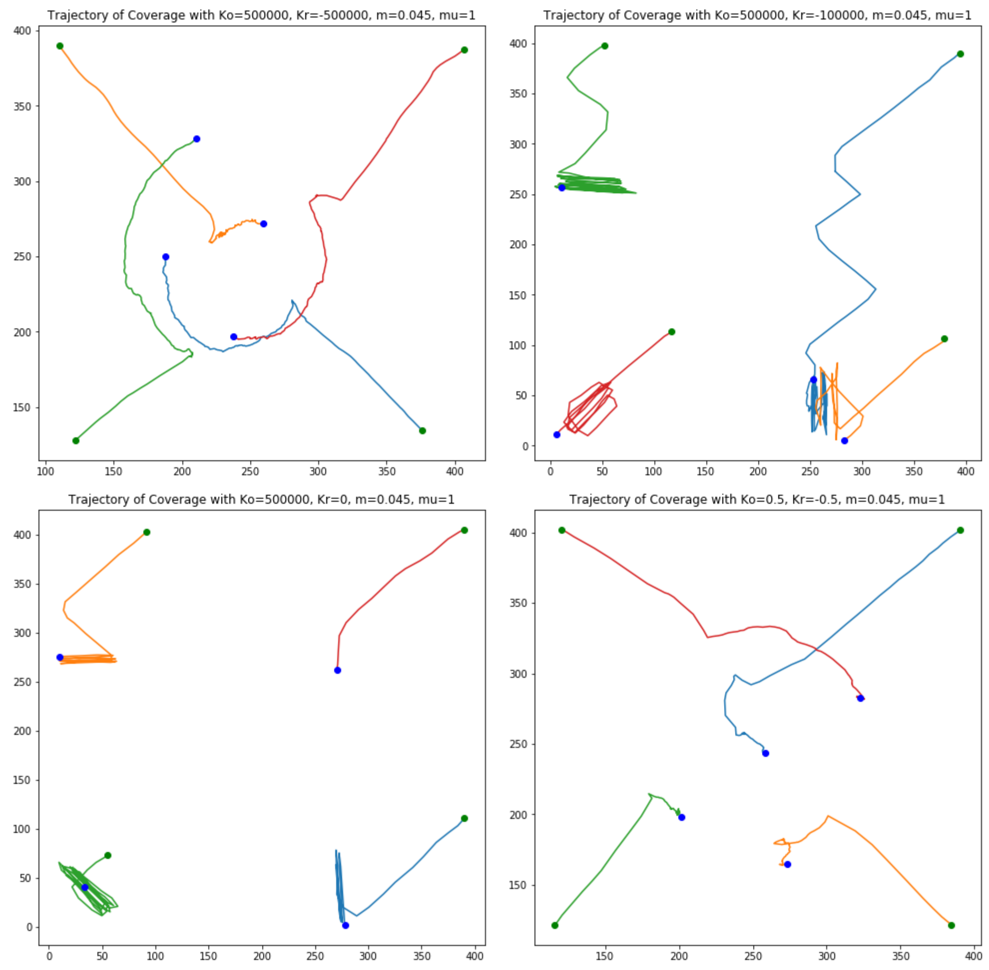

Figure 6: Trajectories of the Attractive Coverage with Different Parameters

In this part, we re-implement the aggregation function of 4 robots by using coverage and attractive potential field. We also plot the trajectories of the 3 or 4 robots by aggregation. It can be observed that the robots has a trend to aggregate together and the aggregation performance can be more smooth than section 3 if the parameters are properly tuned. Different parameters can lead to different performance. According to Figure6, the first and fourth experiment have a smooth trajectory of aggregation.

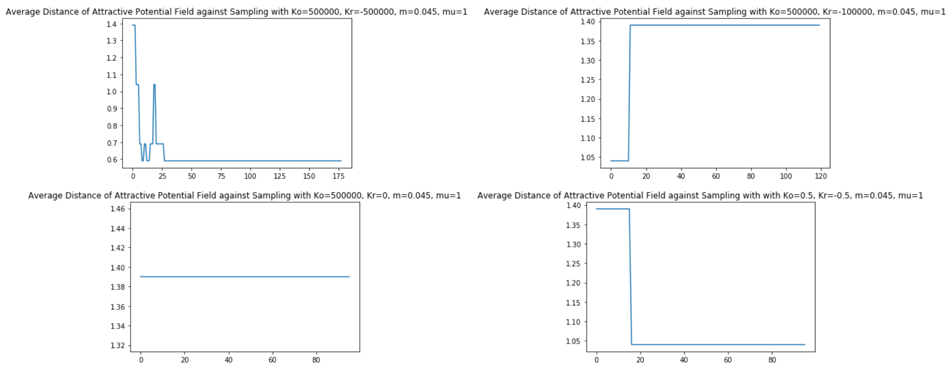

Figure 7: Entropy of the Attractive Coverage with Different Parameters

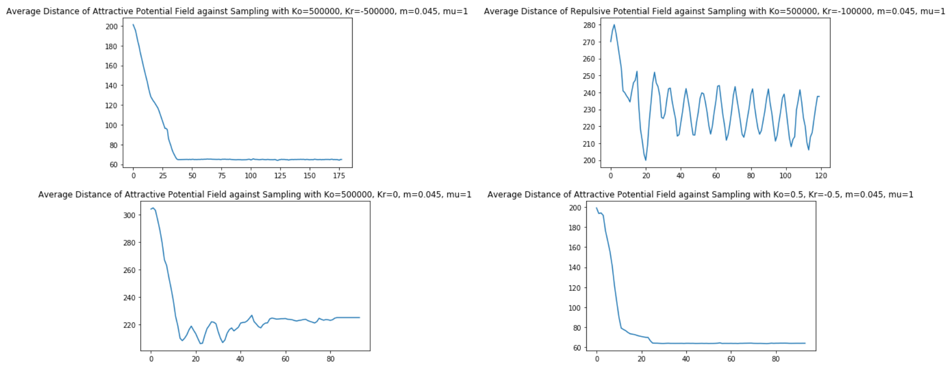

Figure 8: Average Distances of the Attractive Coverage with Different Parameters

The 2 figures above show the metrics of coverage performance. The average distance metric is similar to the total distance metric in section 3 and reduced distance show better aggregation result. Besides, the entropy is also an intuitive metric to evaluate the performance. For example, if 4 robots are located in separate sub-regions the entropy value is highest, reaching around 1.39. If more and more robots get close and get situated in the same sub-region, the entropy value will decrease and the value will be reduced to 0 if all robots are located in one of the 16 sub-regions.

In this section, we mainly evaluate the influence of two gains Ko and Kr in our code, and the sign of 2 parameters are set inverse. From the trajectories and metric plots, we assume that if the absolute values of 2 parameters are properly close, the aggregation performance tends to be better and more obvious. Also, larger value of the gains can be represented as stronger potential field forces and attraction to each other. Thus, the first experiment where the gains are quite high show a stronger aggregation effect of the robots.

Besides, if we invert the sign of the coverage direction in our code, we can also realize the repulsive potential field and the robots can separate apart like they repel each other. The trajectory and performance metric plot is shown below.

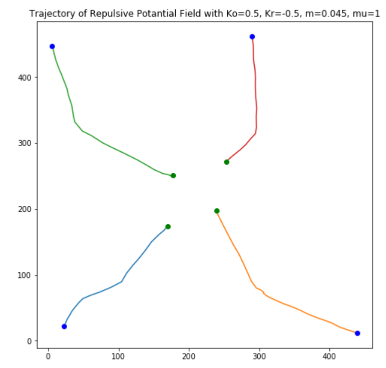

Figure 9: Trajectories of Repulsive Potential Field with Ko=0.5, Kr=-0.5, m=0.045, mu=1

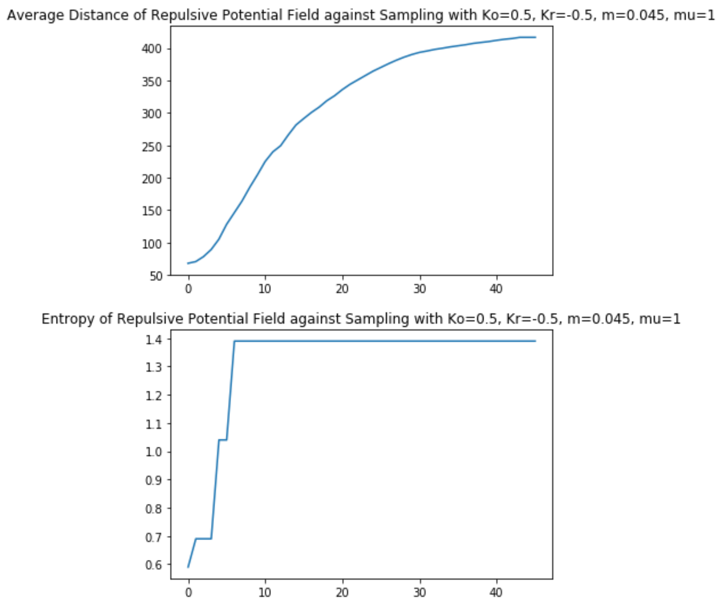

Figure 10: Performance Metrics of Repulsive Potential Field against Sampling with Ko=0.5, Kr=-0.5, m=0.045, mu=1

From Figure 9, we can observe the repulsive reaction among the 4 robots, which can also prove our coverage function is reasonable. Also, the metric of average distance and entropy also show the robots are getting farther and farther from each other. Since the repulsive potential field is strong when the robots are close, after a short time, the robots will be quickly repulsed to 4 different sub-regions, causing the entropy value to be stable around 1.39.

4.3 Conclusion

In this section, we implement a new algorithm to control the cellulo robots by coverage. Upon this, we can realize the aggregation and separation of robots by artificial potential field. After comparison among several parameters (mainly gains) and evaluation by two metrics, we can realize the effective aggregation and separation performance effectively, and it should be noted that the aggregation performance is better and more smooth than the performance in section 3. This method can have great value in real-life applications without sudden fluctuations that may cause dangerous collisions. Due to lack of time, we can implement more precise parameter tuning and try to improve the robustness of the algorithm in the future work.

5 Answers to Deliverables

5.1 Deliverable 1

Implement the function "topicCallback getTouchKeys" in the file "ros cellulo interaction ros cellulo leader node.cpp"

```
    void topicCallback_getTouchKeys(const ros_cellulo_swarm::
                                         cellulo_touch_key& message)
    {
        // 1- Evaluate if the call back is a touch or release
        // 2- If it is a touch:
        // a- detect which robot was selected.
        //    i- publish its mac address
        //   ii- turn its leds to red.
        // b- turn the leds of other robots to green
        std_msgs::String mac;
        mac.data = message.header.frame_id.c_str();
        LeaderPublisher.publish(mac);
        for(int i = 0; i < nb_robots; i++){
            ros_cellulo_swarm::cellulo_visual_effect light_cellulo;
            if(present_robots[i+1] == mac.data){
                light_cellulo.red = 255;
                light_cellulo.green = 0;
                light_cellulo.blue = 0;
            }
            else{
                light_cellulo.red = 0;
                light_cellulo.green = 255;
                light_cellulo.blue = 0;
            }
            VisualEffectPublisher[i].publish(light_cellulo);
        }
    }
};
```

5.2 Deliverable 2

Implement the function "followingLeader" in the file "ros cellulo interaction ros cellulo interaction node.cpp"

```
    void followingMyLeader()
    {
        // Implement here your control.
        // 1- Calculate the required velocity
        // (Useful variables: Ku, distance_to_leader and
                                            reference_distance)
        // 2- Publish the velocity
        float vel_x;
        float vel_y;
        vel_x = Ku*(distance_to_leader.x - reference_distance.x);
        vel_y = Ku*(distance_to_leader.y - reference_distance.y);
        geometry_msgs::Vector3 vel_cellulo;
        vel_cellulo.x= vel_x;
        vel_cellulo.y= vel_y;
        vel_cellulo.z = 0.0;

        VelocityPublisher.publish(vel_cellulo);

    }
};
```

5.3 Deliverable 3

Evaluate your implementation by plotting and comparing the trajectories of the leader and follower.

Described in section 2.2.

5.4 Deliverable 4

Implement the two basic behaviours wait and random_walk.

Based on these two behaviors, implement the function "RosCelluloAggregation::naive calculate new velocities()" in the file "src ros cellulo aggregation RosCelluloAggregation.cpp"

```
void RosCelluloAggregation::naive_calculate_new_velocities()
{
    time = time + 1;
    if(nb_of_robots_detected!=-1 && nb_of_obstacles_detected!=-1 &&
                                        velocity_updated)
    {
        //*************************
        // FILL HERE - Part III Step 1
        //*************************
        double velocity_ = 80;
        geometry_msgs::Vector3 vel_cellulo;
        //std::cout<<(rand()%360)<<"endl";
        static double yaw_angle = 0;//((double)(rand()%360)/180.0)*M_PI
                                            ;
        if(nb_of_robots_detected >0){
            //if it sees another robot close by a certain distanceth,
                                                            it waits
            vel_cellulo.x=0;
            vel_cellulo.y=0;
            vel_cellulo.z=0;
        }
        // if wait too longer
        else if(time % 40 == 0){
            //srand(1);
            yaw_angle = ((double)(rand()%360)/180.0)*M_PI;
            //for(int i=0; i<30;i++){
            //yaw_angle = (rand()%(int(360/10)))*10;
            vel_cellulo.x=velocity_*cos(yaw_angle);
            vel_cellulo.y=velocity_*sin(yaw_angle);
            vel_cellulo.z=0;//}
        }
        else
        {
            //yaw_angle = (rand()%360)/2*3.14159;
            //yaw_angle = (rand()%360)/2*3.14159;
            //for(int i=0; i<30;i++){
            vel_cellulo.x=velocity_*cos(yaw_angle);
            vel_cellulo.y=velocity_*sin(yaw_angle);
            vel_cellulo.z=0;/**/
        }
        //std::cout<<(yaw_angle);



        ROS_INFO("random number %lf velocity %lf %lf",yaw_angle,
                                                    vel_cellulo.x,vel_cellulo.y
                                                    );
        RosCelluloAggregation::publisher_Velocity.publish(vel_cellulo);

        nb_of_robots_detected=-1;
        nb_of_obstacles_detected=-1;
        velocity_updated=false;

    }
    else
    {
        return;
    }
}
```

5.5 Deliverable 5

Test your implementation with and increasing number of robots.
What parameters could influence this type of aggregation method?
How does the time of the task completion change?

Try to vary some parameters (sensor threshold, other you might implement) and plot the evaluation metrics against them.

Analysis described in section 3.2.

5.6 Deliverable 6

Derive the force equation $\overrightarrow{F_o}$ in function of k_o and $\overrightarrow{r_i}$ .Where k_o is a constant describing the strength of the field.

Let $\overrightarrow{p} = (x, y)$ denote the position of the robot and let $\overrightarrow{p_i} = (x_i, y_i)$ denote the position of obstacle i (We think that for every fixed i, pi is a constant vector. i.e. the position of the obstacle is unchanged.). So $\overrightarrow{r_i}$ is the vector from position of the robot to the position of the obstacl i. We have:

$$\overrightarrow{r_i} = \overrightarrow{p_i} - \overrightarrow{p} \tag{1}$$

And r_i is the Euclidean distance between the robot and obstacle i.The distance r_i is then given by:

$$r_i = \|\overrightarrow{r_i}\| = \|\overrightarrow{p_i} - \overrightarrow{p}\| \tag{2}$$

So for the "electrostatic" potential:

$$U_o = k_o \sum_i \frac{1}{r_i} = k_o \sum_i \frac{1}{\|\overrightarrow{r_i}\|} = k_o \sum_i \frac{1}{\|\overrightarrow{p_i} - \overrightarrow{p}\|} \tag{3}$$

Using these definitions, the force F_o can be computed as:

$$\overrightarrow{F_o} = -\frac{dU_o}{d\overrightarrow{p}} = -\frac{dU_o}{d(\overrightarrow{p_i} - \overrightarrow{r_i})} = \frac{dU_o}{d(\overrightarrow{r_i} - \overrightarrow{p_i})} = \frac{d\left(k_o \sum_i \frac{1}{\|\overrightarrow{r_i}\|}\right)}{d(\overrightarrow{r_i} - \overrightarrow{p_i})} = -k_o \sum_i \frac{1}{\|\overrightarrow{r_i}\|^2} \tag{4}$$

5.7 Deliverable 7

Derive the control law: what should be the new commanded velocity vector given the total force acting on the robot and its current velocity? Firstly we convert this question to the discrete form. We have: (n=1,2,3....)

$$\begin{cases} \triangle t_n = t_{n+1} - t_n \\ \triangle v_n = v_{n+1} - v_n \\ \triangle x_n = x_{n+1} - x_n \end{cases}$$

And then we can get:

$$\begin{cases} \dot{x}_n = v_n = \frac{\triangle x_n}{\triangle t_n} \\ \ddot{x}_n = \dot{v}_n = a = \frac{\triangle v_n}{\triangle t_n} \\ \ddot{x}_n = \frac{F - \mu \dot{x}_n}{m} \end{cases}$$

So we have:

$$\triangle v_n = \ddot{x}_n \cdot \triangle t_n = \frac{F - \mu v_n}{m} \triangle t_n \tag{5}$$

$$v_{n+1} = v_n + \triangle v_n = v_n + \frac{F - \mu v_n}{m} \cdot \triangle t_n = \frac{F \cdot \triangle t}{m} + \left(\frac{m - \mu \triangle t_n}{m} \right) \cdot v_n \tag{6}$$

where μ is the viscosity coefficient, m is the mass of the robot.

5.8 Deliverable 8

Implement the function "RosCelluloCoverage::calculate_new_velocities" in the file "src/ros_cellulo_coverage/RosCelluloCoverage.cpp."

```
void RosCelluloCoverage::calculate_new_velocities()
{
    //// TO IMPLEMENT ////
    if(nb_of_robots_detected!=-1 && nb_of_obstacles_detected!=-1 &&
                                        velocity_updated)
    {
        float Fox=0;
        float Foy = 0;
        float Fnx = 0;
        float Fny = 0;
        float Vx = 0;
        float Vy = 0;

        for(int i=0;i<nb_of_obstacles_detected;i++)
        {
            Fox=Fox+ko*(distances_to_obstacles[i]).x;
            Foy=Foy+ko*(distances_to_obstacles[i]).y;

        }

        for(int i=0;i<nb_of_robots_detected;i++)
        {
            Fnx=Fnx+kr*(distances_to_robots[i].x);
            Fny=Fny+kr*(distances_to_robots[i].y);

        }
        float Fx= Fox+Fnx;// fro attractive you just need to change the
                                                    minus sign
        float Fy = Foy +Fny;

        float t = 1/rate;
        Vx = Vx*(1-float(t)*mu/m) + Fx*t/m;
        Vy = Vy*(1-float(t)*mu/m) + Fy*t/m;

        geometry_msgs::Vector3 vel;
        vel.x = Vx;
        vel.y = Vy;
        vel.z = 0;

        RosCelluloCoverage::publisher_Velocity.publish(vel);


        nb_of_robots_detected=-1;
        nb_of_obstacles_detected=-1;
        velocity_updated=false;
    }

}
```

5.9 Deliverable 9

Study the effect of varying these factors on the quality of coverage measured using the metric

Code for metric:

```
# create 16 regions
# the list contains left down cordinates of each rectangle region
regions=[]
for i in range(4):
    for j in range(4):
        temp_w=i*125
        temp_h=j*125
        regions.append([temp_w,temp_h])

def region_of_one_car(temp_x,temp_y):
    for i in range(16):
        temp_w=regions[i][0]
        temp_h=regions[i][1]
        if temp_x>=temp_w and temp_x<temp_w+125 and temp_y>=temp_h and
                                               temp_y<temp_h+125:
            return i

# calculate the entropy metric
def calculate_entropy(temp_x1,temp_y1,temp_x2,temp_y2, temp_x3,temp_y3,
                                         temp_x4,temp_y4):
    region_ratio=np.zeros(16)
    if True:
        region_rob_1=region_of_one_car(temp_x1,temp_y1)
        region_rob_2=region_of_one_car(temp_x2,temp_y2)
        region_rob_3=region_of_one_car(temp_x3,temp_y3)
        region_rob_4=region_of_one_car(temp_x4,temp_y4)


        for i in range(16):
            if region_rob_1==i:
                region_ratio[i]+=1
            if region_rob_2==i:
                region_ratio[i]+=1
            if region_rob_3==i:
                region_ratio[i]+=1
            if region_rob_4==i:
                region_ratio[i]+=1

        #region_ratio=region_ratio/(125*125)
        region_p=region_ratio/(np.sum(region_ratio))

        entropy=0
        print(region_ratio)
        for i in range(16):
            if region_p[i]==0:
                temp_entropy=0
            else:
                temp_entropy=-region_p[i]*np.log(region_p[i])

            entropy+=temp_entropy
        return entropy

def plot_entropy(x1_list=x1,y1_list=y1,x2_list=x2,y2_list=y2,x3_list=x3
                                        ,y3_list=y3,x4_list=x4,y4_list=y4,
                                        title=''):
    entropy_list=[]
    for i in range(len(x1_list)):
        temp_e=calculate_entropy(x1_list[i],y1_list[i],x2_list[i],
                                               y2_list[i],x3_list[i],
                                               y3_list[i],x4_list[i],
                                               y4_list[i])
        entropy_list.append(temp_e)
    plt.title(title)
    plt.plot(entropy_list)
```

```
def calculate_dist(temp_x1,temp_y1,temp_x2,temp_y2):
    return np.sqrt((temp_x1-temp_x2)*(temp_x1-temp_x2)+(temp_y1-temp_y2
                                        )*(temp_y1-temp_y2))

def plot_avg_dist(x1_list=x1,y1_list=y1,x2_list=x2,y2_list=y2,x3_list=
                                        x3,y3_list=y3,x4_list=x4,y4_list=y4
                                        ,title=''):
    dist_list=[]
    for i in range(len(x1_list)):
        dist_12=calculate_dist(x1_list[i],y1_list[i],x2_list[i],y2_list
                                        [i])
        dist_13=calculate_dist(x1_list[i],y1_list[i],x3_list[i],y3_list
                                        [i])
        dist_14=calculate_dist(x1_list[i],y1_list[i],x4_list[i],y4_list
                                        [i])
        dist_23=calculate_dist(x2_list[i],y2_list[i],x3_list[i],y3_list
                                        [i])
        dist_24=calculate_dist(x2_list[i],y2_list[i],x4_list[i],y4_list
                                        [i])
        dist_34=calculate_dist(x3_list[i],y3_list[i],x4_list[i],y4_list
                                        [i])
        temp_avg_dist=(dist_12+dist_13+dist_14+dist_23+dist_24+dist_34)
                                        /6
        dist_list.append(temp_avg_dist)
    plt.title(title)
    plt.plot(dist_list)
```

Analysis described in section 4.2.

5.10 Deliverable 10

Implement the function RosCelluloAggregation::field_calculate_new_velocities() in the file src/ros_cellulo_aggregation/RosCelluloAggregagtion.cpp.

```
void RosCelluloAggregation::field_calculate_new_velocities()
{

    if(nb_of_robots_detected!=-1 && nb_of_obstacles_detected!=-1 &&
                                        velocity_updated)
    {


        geometry_msgs::Vector3 vel;
        ROS_INFO("velocity calcul %lf %lf",vel.x,vel.y);
        RosCelluloAggregation::publisher_Velocity.publish(vel);

        nb_of_robots_detected=-1;
        nb_of_obstacles_detected=-1;
        velocity_updated=false;

    }
    else
    {
        return;
    }
}
```

5.11 Deliverable 11

Study the effect of varying the physics factors on the aggregation performance metrics.

Described in section 4.2.

6 Conclusion

In this practical, we get familiar with some basic capacities of ROS as well as the cellulo robots. We implement our method of controlling the robots of leader following, aggregation and coverage, and verify the effect of our method by observing the real trajectories of cellulo robots and analyze the performance based on the logs. Also, we try to study different influence of different parameters when implementing our control methods and try to reach a reasonable performance of cellulo robots. In our future work, we will focus on better algorithms with more precise parameter tuning and can take more sensor information into account to interact with environments, thus achieving more advanced functions.

7 Appendix

7.1 Code for Leader Follower

```
#include <ros/ros.h>
#include "ros_cellulo_swarm/cellulo_touch_key.h"
#include "ros_cellulo_swarm/cellulo_visual_effect.h"
#include "std_msgs/String.h"

class LeaderSelection
{
public:
    explicit LeaderSelection(ros::NodeHandle& nodeHandle): nodeHandle_(
                                                    nodeHandle){}

    //Parameters
    std_msgs::String leader;
    char** present_robots; //points to the mac addresses of the robots
                                                    present in the launch file.
                                //Indexing starts at one so present_robots[1
                                                                        ] is the
                                                                         the
                                                                        first
                                                                        robot,
                                                                        present_robots
                                                                        [2] the
                                                                        second,
                                                                        etc ...

    int nb_robots; // indicates the number of robots
```

```
//! ROS node handle.
ros::NodeHandle& nodeHandle_;

//! ROS subscirbers and publishers.
ros::Subscriber* TouchedRobots; //Subscribes to long touch events
ros::Publisher* VisualEffectPublisher; //Publishes visual effects (
                                   leds)
ros::Publisher LeaderPublisher; //Publishes how is the leader.

void setSubscribersPublishers()
{
    TouchedRobots=new ros::Subscriber[nb_robots];
    VisualEffectPublisher=new ros::Publisher[nb_robots];

    for(int i=0; i<nb_robots;i++)
    {
        char subscriberTopic[100];
        sprintf(subscriberTopic, "/cellulo_node_%s/longTouchKey",
                                       present_robots[i+1]);
        TouchedRobots[i]=nodeHandle_.subscribe(subscriberTopic, 10,
                                       &LeaderSelection::
                                       topicCallback_getTouchKeys
                                       ,this);

        //Publishers
        char publisherTopic[100];
        sprintf(publisherTopic, "/cellulo_node_%s/setVisualEffect",
                                       present_robots[i+1]);
        VisualEffectPublisher[i] = nodeHandle_.advertise<
                                       ros_cellulo_swarm::
                                       cellulo_visual_effect>(
                                       publisherTopic,10);
    }
    LeaderPublisher=nodeHandle_.advertise<std_msgs::String>("/
                                   leader",1);
}

// Call back function if a change on one of the long touch sensors
                                   on one of the robots is
                                   detected.
// The functions should detects which robot was touched and publish
                                    its mac_adress on the
                                   LeaderPublisher
void topicCallback_getTouchKeys(const ros_cellulo_swarm::
                                   cellulo_touch_key& message)
{
    // 1- Evaluate if the call back is a touch or release
    // 2- If it is a touch:
    // a- detect which robot was selected.
    //    i- publish its mac address
    //   ii- turn its leds to red.
```

```
        // b- turn the leds of other robots to green
        std_msgs::String mac;
        mac.data = message.header.frame_id.c_str();
        LeaderPublisher.publish(mac);
        for(int i = 0; i < nb_robots; i++){
            ros_cellulo_swarm::cellulo_visual_effect light_cellulo;
            if(present_robots[i+1] == mac.data){
                light_cellulo.red = 255;
                light_cellulo.green = 0;
                light_cellulo.blue = 0;
            }
            else{
                light_cellulo.red = 0;
                light_cellulo.green = 255;
                light_cellulo.blue = 0;
            }
            VisualEffectPublisher[i].publish(light_cellulo);
        }
    }
};

int main(int argc, char** argv)
{
    ros::init(argc, argv, "ros_cellulo_leader");
    ros::NodeHandle nodeHandle("~");

    LeaderSelection leaderSelection(nodeHandle);

    leaderSelection.present_robots=argv;
    leaderSelection.nb_robots=argc-1;

    leaderSelection.setSubscribersPublishers();
    ros::spin();

    return 0;
}
```

```
#include <ros/ros.h>
#include "ros_cellulo_swarm/cellulo_touch_key.h"
#include "ros_cellulo_swarm/cellulo_visual_effect.h"
#include "ros_cellulo_swarm/ros_cellulo_sensor.h"
#include "std_msgs/String.h"
#include "geometry_msgs/Pose2D.h"
#include "tf2_ros/transform_listener.h"
#include "std_msgs/Float64.h"


class FollowLeader
{
public:
    explicit FollowLeader(ros::NodeHandle& nodeHandle): nodeHandle_(
                                                nodeHandle){
```

```
        if (!readParameters()) {

                ROS_ERROR("Could not read parameters :(.");
                ros::requestShutdown();
        }
        subscriber_setKu = nodeHandle_.subscribe("/setKu", 1, &
                                    FollowLeader::
                                    topicCallback_setKu, this);
        LeaderSubscriber=nodeHandle_.subscribe("/leader",10,&
                                    FollowLeader::
                                    leader_callback,this);
        leader_selected=false;
    }

    //Parameters
    std::string myleader;
    std::string myname;
    bool cleared;
    bool leader_selected=false;
    geometry_msgs::Vector3 distance_to_leader;
    geometry_msgs::Vector3 reference_distance; //the reference distance
                                  set to the distance from
                                  leader in the callback function
                                  of the subscriber to the "
                                  Leader" topic

    //Control parameters
    double Ku;

    //! ROS node handle.
    ros::NodeHandle& nodeHandle_;

    //! Publishers and Subsribers
    ros::Subscriber LeaderSubscriber;
    ros::Subscriber subscriber_Robots;
    ros::Subscriber subscriber_setKu;
    ros::Publisher VelocityPublisher;
    ros::Publisher clearTrackingPublisher;


    bool readParameters()
    {
            if (!nodeHandle_.getParam("Ku", Ku))
                return false;

            return true;
    }
    void setSubscribersPublishers()
    {
        char subscriberTopicRobots[100];
        sprintf(subscriberTopicRobots, "/sensor_node_%s/detectedRobots"
                                    ,myname.c_str());
        subscriber_Robots=nodeHandle_.subscribe(subscriberTopicRobots,1
                                                ,&FollowLeader::
                                                topicCallback_getDetectedRobots
                                                ,this);

        char publisherTopic_setvelocity[100];
        sprintf(publisherTopic_setvelocity, "/cellulo_node_%s/
                                                setGoalVelocity",myname.
                                                c_str());
        VelocityPublisher = nodeHandle_.advertise<geometry_msgs::
                                                Vector3>(
                                                publisherTopic_setvelocity,
                                                1);

        char publisherTopic_clearTracking[100];
        sprintf(publisherTopic_clearTracking, "/cellulo_node_%s/
                                                clearTracking",myname.c_str
                                                ());
        clearTrackingPublisher = nodeHandle_.advertise<std_msgs::Empty>
                                                (
                                                publisherTopic_clearTracking
                                                ,1);
    }

    void leader_callback(std_msgs::String message)
    {
        myleader=message.data;
        cleared=false;
        leader_selected=true;
        reference_distance=distance_to_leader;

    }
    void topicCallback_setKu(const std_msgs::Float64 & message)
    {
        Ku=message.data;
    }
    void clearTracking(){
        std_msgs::Empty clear;
        clearTrackingPublisher.publish(clear);
        cleared=true;

    }

    void topicCallback_getDetectedRobots(ros_cellulo_swarm::
                                        ros_cellulo_sensor sensor)
    {
        if(sensor.detected!=0)
            distance_to_leader = sensor.Distance[0];
    }

    void followingMyLeader()
    {
        // Implement here your control.
        // 1- Calculate the required velocity
        // (Useful variables: Ku, distance_to_leader and
                                              reference_distance)
        // 2- Publish the velocity
        float vel_x;
        float vel_y;
        vel_x = Ku*(distance_to_leader.x - reference_distance.x);
        vel_y = Ku*(distance_to_leader.y - reference_distance.y);
        geometry_msgs::Vector3 vel_cellulo;
        vel_cellulo.x= vel_x;
        vel_cellulo.y= vel_y;
        vel_cellulo.z = 0.0;

        VelocityPublisher.publish(vel_cellulo);

    }
};

int main(int argc, char** argv)
{
    ros::init(argc, argv, "ros_cellulo_interaction");
    ros::NodeHandle nodeHandle("~");

    FollowLeader followLeader(nodeHandle);

    followLeader.myname=argv[1];
    followLeader.setSubscribersPublishers();
    ros::Rate rate(20);
    while(ros::ok())
    {
        if(followLeader.leader_selected)
        {
            if(followLeader.myleader!=followLeader.myname)
                followLeader.followingMyLeader();
            else {
                if(!followLeader.cleared)
                    followLeader.clearTracking();
            }
        }

        ros::spinOnce();
        rate.sleep();
    }

    return 0;
}
```

7.2 Code for Aggregation

```
#include "ros_cellulo_aggregation/RosCelluloAggregation.hpp"
```

```
#include "tf2_ros/transform_listener.h"

RosCelluloAggregation::RosCelluloAggregation(ros::NodeHandle&
                                    nodeHandle) : nodeHandle_(
                                    nodeHandle)
{
        if (!readParameters()) {

                ROS_ERROR("Could not read parameters :(.");
                ros::requestShutdown();
        }

        //subscribers
  // TODO add subscribers to parameters you want to test in live

        nb_of_robots_detected=-1;
        nb_of_obstacles_detected=-1;
        velocity_updated=false;
        time=0;

}

RosCelluloAggregation::~RosCelluloAggregation(){
}

bool RosCelluloAggregation::readParameters()
{
         if(!nodeHandle_.getParam("scale", scale))
            return false;
        return true;
}


void RosCelluloAggregation::setPublishers()
{
    //publisher
    char publisherTopic_setvelocity[100];
    sprintf(publisherTopic_setvelocity, "/cellulo_node_%s/
                                        setGoalVelocity",
                                        RosCelluloAggregation::mac_Adr)
                                        ;
    publisher_Velocity = nodeHandle_.advertise<geometry_msgs::Vector3>(
                                        publisherTopic_setvelocity,1);

}

void RosCelluloAggregation::setSubscribers()
{
    //subscribers
    char subscriberTopicRobots[100],subscriberTopicObstacles[100],
                                        subscriberTopicVelocity[100];
    sprintf(subscriberTopicRobots, "/sensor_node_%s/detectedRobots",
                                        mac_Adr);

    sprintf(subscriberTopicObstacles, "/sensor_node_%s/
                                        detectedObstacles",mac_Adr);
    subscriber_Robots=nodeHandle_.subscribe(subscriberTopicRobots,1,&
                                        RosCelluloAggregation::
                                        topicCallback_getDetectedRobots
                                        ,this);
    subscriber_Obstacles=nodeHandle_.subscribe(subscriberTopicObstacles
                                        ,1,&RosCelluloAggregation::
                                        topicCallback_getDetectedObstacles
                                        ,this);

    sprintf(subscriberTopicVelocity, "/cellulo_node_%s/velocity",
                                        mac_Adr);
    subscriber_Velocity= nodeHandle_.subscribe(subscriberTopicVelocity,
                                     1, &RosCelluloAggregation::
                                        topicCallback_getVelocity,this)
                                        ;

}

void RosCelluloAggregation::topicCallback_getDetectedRobots(
                                    ros_cellulo_swarm::
                                    ros_cellulo_sensor sensor)
{
    distances_to_robots=sensor.Distance;
    nb_of_robots_detected=sensor.detected;
}

void RosCelluloAggregation::topicCallback_getDetectedObstacles(
                                    ros_cellulo_swarm::
                                    ros_cellulo_sensor sensor)
{
    distances_to_obstacles=sensor.Distance;
    nb_of_obstacles_detected=sensor.detected;

}

void RosCelluloAggregation::topicCallback_getVelocity(geometry_msgs::
                                    Vector3 v)
{
    RosCelluloAggregation::velocity.x=v.x;
    RosCelluloAggregation::velocity.y=v.y;
    RosCelluloAggregation::velocity_updated=true;

}

void RosCelluloAggregation::naive_calculate_new_velocities()
{
    time = time + 1;
```

```
if(nb_of_robots_detected!=-1 && nb_of_obstacles_detected!=-1 &&
                                          velocity_updated)
{
    //***************************
    // FILL HERE - Part III Step 1
    //***************************
    double velocity_ = 80;
    geometry_msgs::Vector3 vel_cellulo;
    //std::cout<<(rand()%360)<<"endl";
    static double yaw_angle = 0;//((double)(rand()%360)/180.0)*M_PI
                                               ;
    if(nb_of_robots_detected >0){
        //if it sees another robot close by a certain distanceth,
                                                      it waits
        vel_cellulo.x=0;
        vel_cellulo.y=0;
        vel_cellulo.z=0;
    }
    // if wait too longer
    else if(time % 40 == 0){
        //srand(1);
        yaw_angle = ((double)(rand()%360)/180.0)*M_PI;
        //for(int i=0; i<30;i++){
        //yaw_angle = (rand()%(int(360/10)))*10;
        vel_cellulo.x=velocity_*cos(yaw_angle);
        vel_cellulo.y=velocity_*sin(yaw_angle);
        vel_cellulo.z=0;//}
    }
    else
    {
        //yaw_angle = (rand()%360)/2*3.14159;
        //yaw_angle = (rand()%360)/2*3.14159;
        //for(int i=0; i<30;i++){
        vel_cellulo.x=velocity_*cos(yaw_angle);
        vel_cellulo.y=velocity_*sin(yaw_angle);
        vel_cellulo.z=0;/**/
    }
    //std::cout<<(yaw_angle);



    ROS_INFO("random number %lf velocity %lf %lf",yaw_angle,
                                               vel_cellulo.x,vel_cellulo.y
                                               );
    RosCelluloAggregation::publisher_Velocity.publish(vel_cellulo);

    nb_of_robots_detected=-1;
    nb_of_obstacles_detected=-1;
    velocity_updated=false;

}
else
```

```
    {
        return;
    }
}


void RosCelluloAggregation::field_calculate_new_velocities()
{

    if(nb_of_robots_detected!=-1 && nb_of_obstacles_detected!=-1 &&
                                            velocity_updated)
    {


        geometry_msgs::Vector3 vel;
        ROS_INFO("velocity calcul %lf %lf",vel.x,vel.y);
        RosCelluloAggregation::publisher_Velocity.publish(vel);

        nb_of_robots_detected=-1;
        nb_of_obstacles_detected=-1;
        velocity_updated=false;

    }
    else
    {
        return;
    }
}



double RosCelluloAggregation::norm(double x,double y)
{
        return sqrt(pow(x,2)+pow(y,2));
}
```

7.3 Code for Coverage

```
#include "ros_cellulo_coverage/RosCelluloCoverage.hpp"
#include "tf2_ros/transform_listener.h"

RosCelluloCoverage::RosCelluloCoverage(ros::NodeHandle& nodeHandle) :
                                    nodeHandle_(nodeHandle)
{
        if (!readParameters()) {

                ROS_ERROR("Could not read parameters :(.");
                ros::requestShutdown();
        }
        //subscribers
        subscriber_setMass = nodeHandle_.subscribe(
                                             subscriberTopic_setMass, 1,
                                              &RosCelluloCoverage::
                                             topicCallback_setMass, this
                                             );
        subscriber_setViscocity=nodeHandle_.subscribe(
                                             subscriberTopic_setViscocity
                                             ,1,&RosCelluloCoverage::
                                             topicCallback_setViscocity,
                                             this);
        subscriber_setFieldStrengthObstacles=nodeHandle_.subscribe(
                                             subscriberTopic_setFieldStrengthObstacle
                                             ,1,&RosCelluloCoverage::
                                             topicCallback_setFieldStrengthObstacles
                                             ,this);
        subscriber_setFieldStrengthRobots=nodeHandle_.subscribe(
                                             subscriberTopic_setFieldStrengthRobots
                                             ,1,&RosCelluloCoverage::
                                             topicCallback_setFieldStrengthRobots
                                             ,this);

        nb_of_robots_detected=-1;
        nb_of_obstacles_detected=-1;
        velocity_updated=false;

}

RosCelluloCoverage::~RosCelluloCoverage(){
}

bool RosCelluloCoverage::readParameters()
{
        if (!nodeHandle_.getParam("ko",ko))
            return false;
        else if(!nodeHandle_.getParam("kr", kr))
            return false;
        else if(!nodeHandle_.getParam("m", m))
            return false;
        else if(!nodeHandle_.getParam("mu", mu))
            return false;
        else if(!nodeHandle_.getParam("scale", scale))
            return false;

        return true;
}

void RosCelluloCoverage::topicCallback_setMass(const std_msgs::Float64
                                     & message)
{
        m=message.data;
}
```

```
void RosCelluloCoverage::topicCallback_setViscocity(const std_msgs::
                                    Float64 &message)
{
        mu=message.data;
}

void RosCelluloCoverage::topicCallback_setFieldStrengthObstacles(const
                                    std_msgs::Float64 &message)
{
        ko=message.data;
}

void RosCelluloCoverage::topicCallback_setFieldStrengthRobots(const
                                    std_msgs::Float64 &k)
{
        kr=k.data;
}

void RosCelluloCoverage::setPublishers()
{
    //publisher
    char publisherTopic_setvelocity[100];
    sprintf(publisherTopic_setvelocity, "/cellulo_node_%s/
                                        setGoalVelocity",
                                        RosCelluloCoverage::mac_Adr);
    publisher_Velocity = nodeHandle_.advertise<geometry_msgs::Vector3>(
                                        publisherTopic_setvelocity,1);

}

void RosCelluloCoverage::setSubscribers()
{
    //subscribers
    char subscriberTopicRobots[100],subscriberTopicObstacles[100],
                                        subscriberTopicVelocity[100];
    sprintf(subscriberTopicRobots, "/sensor_node_%s/detectedRobots",
                                        mac_Adr);
    //ROS_INFO("=====-%s",subscriberTopicRobots);
    sprintf(subscriberTopicObstacles, "/sensor_node_%s/
                                        detectedObstacles",mac_Adr);
    subscriber_Robots=nodeHandle_.subscribe(subscriberTopicRobots,1,&
                                        RosCelluloCoverage::
                                        topicCallback_getDetectedRobots
                                        ,this);
    subscriber_Obstacles=nodeHandle_.subscribe(subscriberTopicObstacles
                                        ,1,&RosCelluloCoverage::
                                        topicCallback_getDetectedObstacles
                                        ,this);

    sprintf(subscriberTopicVelocity, "/cellulo_node_%s/velocity",
                                        mac_Adr);
    subscriber_Velocity= nodeHandle_.subscribe(subscriberTopicVelocity,
                                         1, &RosCelluloCoverage::
                                        topicCallback_getVelocity,this)
                                        ;

}

void RosCelluloCoverage::topicCallback_getDetectedRobots(
                                    ros_cellulo_swarm::
                                    ros_cellulo_sensor sensor)
{
    distances_to_robots=sensor.Distance;
    nb_of_robots_detected=sensor.detected;
}

void RosCelluloCoverage::topicCallback_getDetectedObstacles(
                                    ros_cellulo_swarm::
                                    ros_cellulo_sensor sensor)
{
    distances_to_obstacles=sensor.Distance;
    nb_of_obstacles_detected=sensor.detected;
}

void RosCelluloCoverage::topicCallback_getVelocity(geometry_msgs::
                                    Vector3 v)
{
    RosCelluloCoverage::velocity.x=v.x;
    RosCelluloCoverage::velocity.y=v.y;
    RosCelluloCoverage::velocity_updated=true;

}

void RosCelluloCoverage::calculate_new_velocities()
{
    //// TO IMPLEMENT ////
    if(nb_of_robots_detected!=-1 && nb_of_obstacles_detected!=-1 &&
                                        velocity_updated)
    {
        float Fox=0;
        float Foy = 0;
        float Fnx = 0;
        float Fny = 0;
        float Vx = 0;
        float Vy = 0;

        for(int i=0;i<nb_of_obstacles_detected;i++)
        {
            Fox=Fox+ko*(distances_to_obstacles[i]).x;
            Foy=Foy+ko*(distances_to_obstacles[i]).y;

        }
        for(int i=0;i<nb_of_robots_detected;i++)
        {
            Fnx=Fnx+kr*(distances_to_robots[i].x);
            Fny=Fny+kr*(distances_to_robots[i].y);

        }

        float Fx= Fox+Fnx;// fro attractive you just need to change the
                                                     minus sign
        float Fy = Foy +Fny;

        float t = 1/rate;
        Vx = Vx*(1-float(t)*mu/m) + Fx*t/m;
        Vy = Vy*(1-float(t)*mu/m) + Fy*t/m;

        geometry_msgs::Vector3 vel;
        vel.x = Vx;
        vel.y = Vy;
        vel.z = 0;

        RosCelluloCoverage::publisher_Velocity.publish(vel);


        nb_of_robots_detected=-1;
        nb_of_obstacles_detected=-1;
        velocity_updated=false;
    }

}


geometry_msgs::Vector3 RosCelluloCoverage::limit_velocity(geometry_msgs
                                          ::Vector3 v,double limit)
{
    // TO DO //
    //Limit speed to a maximum (300 mm/s). NB: direction should not
                                        change, only the norm.
    //You can also put a lower bound for the speed to avoid small in-
                                        place oscillations
    geometry_msgs::Vector3 newV;
    return newV;
}

double RosCelluloCoverage::norm(double x,double y)
{
        return sqrt(pow(x,2)+pow(y,2));
}
```