Abhijeet Mishra

UNI-aam2285

Evolutionary Computation and Design Automation

Assignment3

Phase B

MECS 4510

Instructor-Hod Lipson

Date Submitted: Dec 5

Grace Hours Accumulated:120 hours

Grace Hours Remaining: 3 hours

In collaboration with

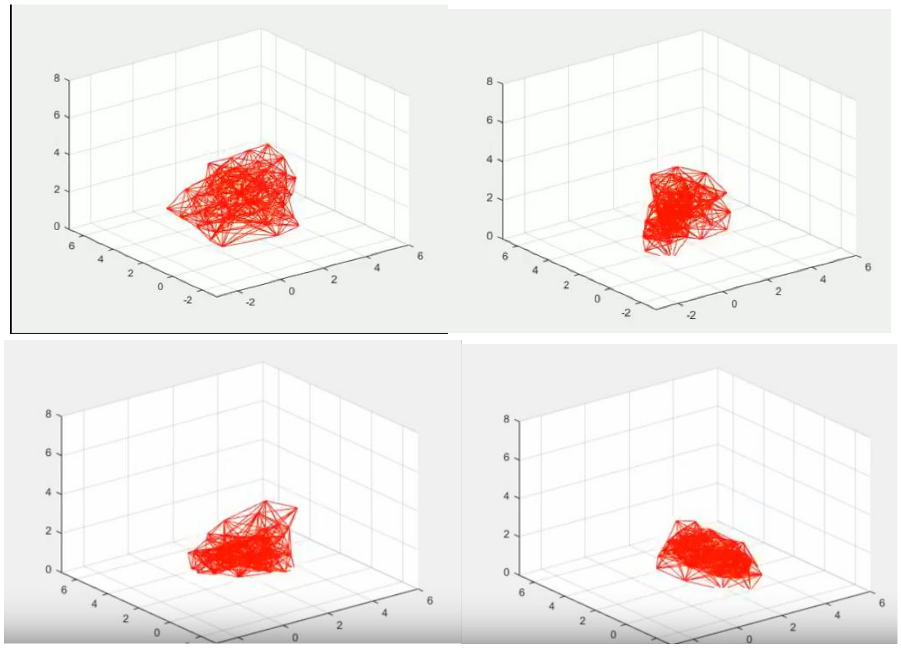
Shivani Vartak

(sv2566)

Grace Hours Accumulated: 120 hours

Grace Hours Remaining: hours

Result



Best Cube Bouncing: https://www.youtube.com/watch?v=XsSJZ19FRIk

Free Block Bouncing: https://www.youtube.com/watch?v=ajE-r0Hv3X4

Fastest Cube Bouncing: https://www.youtube.com/watch?v=4hL6zJBd6UU

Failed Cube 1: https://www.youtube.com/watch?v=kZ-XLruWJA4

Failed Cube 2: https://www.youtube.com/watch?v=eCKaGnVbbDM

Failed Cube 3: https://www.youtube.com/watch?v=1lUK6-NMr8A

Final Cube 1: https://www.youtube.com/watch?v=Gvzgwt_c7Is

Final Cube 2: https://www.youtube.com/watch?v=xTFWBJexo2Q

Final Cube 3: https://www.youtube.com/watch?v=fnmM9A7mqRE

Methods

Parameters Used for Actuation Pattern

The actuation of the soft robot is done by varying the rest length of the robot as a function of time. A larger cube consisting of 3x3x3 smaller cubes is used as the specimen. The initial length of springs on the edge of each smaller cube is taken as 1. We evolve the actuation of this larger cube by changing the parameters which affect the rest length. The equation used for updated rest length is $L=L_0*(a+b\sin(wt+c))$. The spring force is calculated varying the spring constant. The parameters thus varied are b,c and the spring constant(k). The resulting force is calculated by changing the k. The change in rest length also affects this force. Using the force, the acceleration and velocity is calculated which in turn is used to find the displacement. The maximum displacement criteria is used to get the best combination of parameters b, c and k. Here the parameter b is a linear scaling parameter and c is for angular scaling of the sinusoidal function.

The friction coefficient is 0.7. If we increase the co-efficient of friction, then the velocity and acceleration will increase.

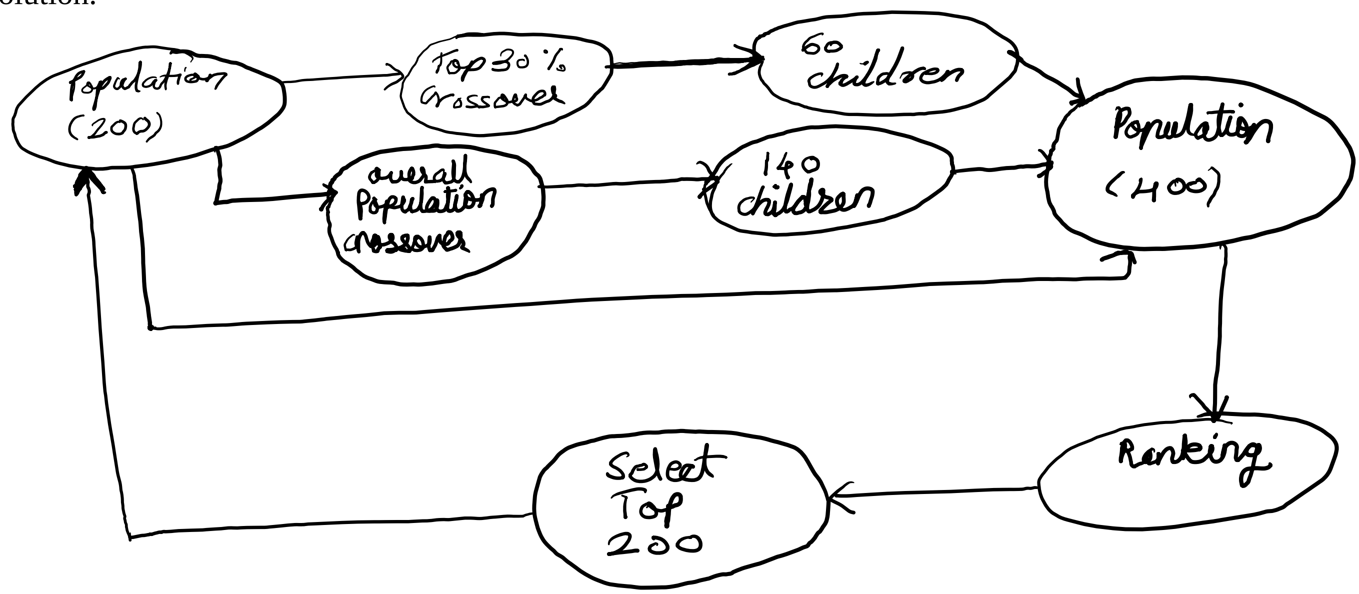
Robot Parameters

The Robot parameters are b, c, k. The initial dimensions of the robot are 3x3x3. Each small cube has a length of 1m, and thus the length of the larger cube is 3m*3m*3m. As k, b, c are varied the robot parameters change. 'k' describes the spring constants of all the springs of a particular small cube. We have taken each small cube as an independent structure and measure it separately. The accelerations on all points of the cube are then added to a master acceleration table which consists of all the points in the mega cube. Here the accelerations of all the points within the single cube are added. Thus this basically acts as a system where if a edge is shared between multiple cubes, we will consider that all springs act there. However multiple spring forces act as a parallel set.

Parameters Used for Evolutionary Process

The evolution of the motion of the soft robot is done by evolving k, b and c. The individual is represented as a single dimensional array of 81 elements where the data of each smaller cube is stored as k, b, c in sequence. Hence, data of each cube exists in 3 elements sequentially and every multiple of 3 starts the representation of a new smaller cube. Every spring in a smaller cube has the same k, b, c. The Evolution is carried out by crossover followed by mutation.

An initial population of 200 individuals is generated by random generation of values for k, b, c. The range of k is 1-5000,b is -2 to 2 and c is -pi to pi. The population of 200 individuals is ranked and the top 30 pairs are crossed over by randomly selecting an index in the representation and swapping the elements following that index among the two individuals. The remaining population is similarly crossed-over with itself and the top 30% individuals. Thus, the probability of crossover is made higher for the top 30% population. After the crossover we get an additional population of 200 children which now adds to the population pool, making a population of 400. This population of 400 is ranked and the top 200 are chosen to make the new population. The new population is then mutated by randomly changing one of the values in the representation array. The same procedure is performed on the newly altered population in loops, thus leading to evolution.



What worked and what didn't?

The crossovers and the mutation worked. The value range of b, when kept from -0.2 to 0.2 worked to evolve the cube easily.

When we tried using the island method to increase the diversity of the population the diversity did not increase. This was caused because of the bias of the selection criteria for the top 30%. The island method works such that, we take the top 100 from two independently evolved populations after 300,000 iterations and rank them together. This becomes our new population set. We then perform crossovers on this population and loop it again. When we tried this on our population set there was a bias for one type of the population that was slightly better than the other. The diversity was further reduced. This also resulted in lack of growth. Another method we tried to increase the diversity was to turn up the mutation each time the diversity of the population fell. However this did not help because the mutated individuals almost always got eliminated immediately.

Even when we kept b-value between [-0.6,0.6], the robot behaved erratically. It had good velocity, but its form was disrupted. This can be seen in the videos provided for those robots. Thus parameter range selection is an extremely important criteria.

Another thing that worked better than though was only slightly mutating the parameters. After the crossover, during mutation, we chose to only vary the parameter by maximum of 10% of it original value. For example, b can mutate to a value in the range [0.9b to 1.1b]. However absolute value must be in the range [-0.2 to 0.2] . This method almost acted like a hillclimber combined with a GA. However, we saw that GA with regular mutations in the range of the full value was slightly better.

Spring Evaluations per second

Calculations/second: 2,430,000

One generation is evaluated every 12 seconds. Each generation has 200 individuals with 27x27 springs each. However, in our calculations, we calculate the force of the spring two times in each cube. For example, between points 1 & 2 , we calculate from 1 to 2 once and then from 2 to 1 also. Thus we multiply by a factor of 2. Each of these springs is evaluated once every 0.01 second therefore it is evaluated 100 times every 1 second. This gives us a total of **2,430,000** spring evaluations per second. However, our actual calculations are much more. This is because, in the evaluation of a generation, we also do crossovers, and sorting.

Alternate representations:

An attempt was made to represent the small cubes as 4 different types of materials. Thus, a cube could either be unactive, soft, medium or hard. Each of these types of materials would have a definite K,b and c value.

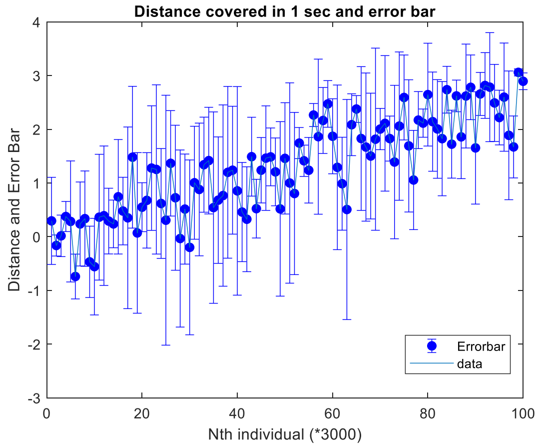
Thus we would evolve a function which basically takes the locations of a particular cube as input and outputs a value between [1,4] which represents the type of material each cube is. If we run this code over all the cubes in the smaller cube, then we can basically represent the parameters of all the small cubes in the entire large cube with the fuction which uses the locations of Centre of Mass of each cube.

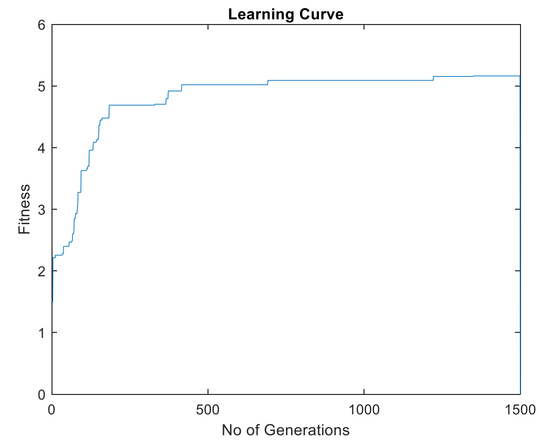
Following this, we would need to evolve the function over several generations.

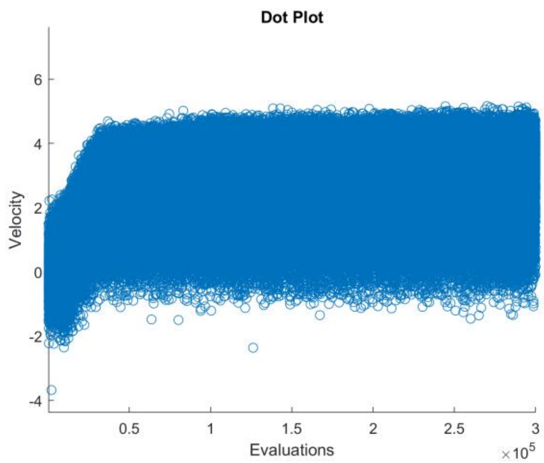
This would be followed by evolving the parameters of each of the materials for the next few materials.

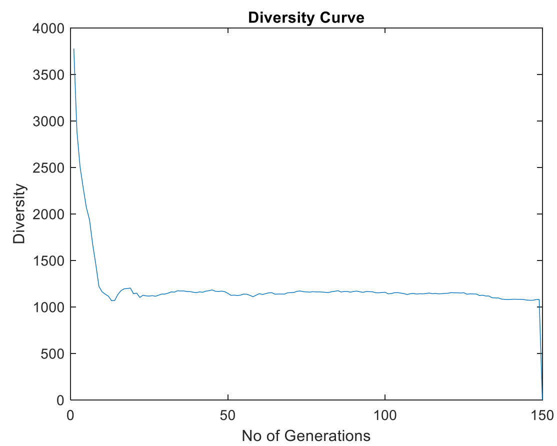
Thus this system would be a small kind of co-evolution where the function to specify the type of material would evolve and also the parameters of the materials would evolve. However this representation would be complicated so we went for a simpler representation.

Performance Plots









The Diversity of the population falls rapidly in the beginning but comes to a critical value and becomes constant. This is because the mutations maintain the diversity as the evaluations increase.

Performance in distance per cycle: 5 distance units/cycle

Code

Plot:

```
clc

%CREATING COM & points for each cube of ALL CUBES
cube_size1 = 3;
length = cube_size1;
width = cube_size1;
height = cube_size1;
COM = cell(length,width,height);
COM_location = zeros(3);
all_points_each_COM = cell(length,width,height);
points_location = zeros(1,3);
initial_height = 0;
pop23 = zeros(10,81);
for k = 1:height
    for j = 1:width
        for i = 1:length
            COM_location = [(0.5 + (i-1)),(0.5 + (j-1)),(0.5 + (k-1))];
            COM{i,j,k} = COM_location;
            pos = 1;
            for z=0:1
                for y = 0:1
                    for x= 0:1
                        points_location(1,pos)= (COM_location(1)+0.5+x);
                        points_location(2,pos)= (COM_location(2)+0.5+y);
                        points_location(3,pos)= (COM_location(3)+0.5+z);
                        pos = pos+1;
                    end
                end
            end
            all_points_each_COM{i,j,k} = points_location;
        end
    end
end
all_points_each_COM{3,3,3};
%COM CREATED
%CREATING POINTS LOCATION FOR EACH CUBE:
acceleration = cell(length+1, width+1, height+1);
vel = cell(length+1, width+1, height+1);
ini_location = cell(length+1,width+1,height+1);
point_location = zeros(3,1);
for k = 0:height
    for j = 0:width
        for i =0:length
            point_location(:,1) = [i;j;k+initial_height];
            ini_location{i+1,j+1,k+1} = point_location;
        end
    end
end
updated_locations = ini_location;
%Index and location for each point created. This includes the location and
%updated location of each point
ideal_dis = cell(length,width,height);
```

```
distance = zeros(8,8);
to_calc=zeros(3,8);
for k=1:height
    for j = 1:width
        for i = 1:length
            for row = 1:8
                l = all_points_each_COM{i,j,k}(1,row);
                m = all_points_each_COM{i,j,k}(2,row);
                n = all_points_each_COM{i,j,k}(3,row);
                %to_calc(:,row)=updated_locations{l,m,n};
                to_calc(1,row)=updated_locations{l,m,n}(1);
                to_calc(2,row)=updated_locations{l,m,n}(2);
                to_calc(3,row)=updated_locations{l,m,n}(3);
            end
            location = to_calc;
            for q=1:8
                for r = 1:q
                    distance(r,q) = sqrt(((location(1,q)-location(1,r))^2)+((location(2,q)-
location(2,r))^2)+((location(3,q)-location(3,r))^2));
                end
            end
            distance = triu(distance)+triu(distance,1)';
            ideal_dis{i,j,k} = distance;
        end
    end
end
ideals = zeros(8,8);
ideals = ideal_dis{1,1,1};
k_ground=10000;

%IDEAL_DISTANCES of all CENTRE of Masses has been calculated

%To calculate the acceleration of all points:
updated_locations = ini_location;
to_calc = zeros(3,8);
ideals = zeros(8,8);
acc = zeros(3,8);

for r = 1:height+1
    for q = 1: width+1
        for p = 1:length+1
            acceleration{p,q,r} = [0;0;0];
            vel{p,q,r} = [0;0;0];
        end
    end
end
total_mega_cubes=10;
size1 = 200;
segmenting =1
itermax = 10000;
plot_diversity = zeros(round(itermax/(size1*10)) , 1);
plot_dot_plot= zeros(itermax,1);
plot_learning = zeros(round(itermax/size1),1);
mut=1;
%%%%INITIALIZING DONE
k = zeros(1,81);
size1 = 10;
population = zeros(size1,81);
```

```
random_k = 0 + (5000);

random_b = 0;

random_c = 0;
for i = 1:27
    population(:,1+3*(i-1)) =  random_k;
    population(:,2+3*(i-1)) =  random_b;
    population(:,3+3*(i-1)) =  random_c;
end
rank=zeros(size1,2);
rank2 = zeros(size1,2);
rank_global = zeros(size1*2,2);
population2 = zeros(size1,81);
pop_global = zeros(size1*2,81);
pop_all = zeros(size1*2,83);
compo = zeros(size1,1);
travel = zeros(size1,1);
t_max = 10*20;
final = cell(10,t_max);
tic
for i=3:3
    %k = pop_ulti_final(i,:);
    k = plot_ulti(i,:);
    updated_locations = ini_location;
    compo(i) = i;
    final = get_final_points2(to_calc,
all_points_each_COM,ideal_dis,updated_locations,ideals,acceleration,vel,k,final,i);
end
toc
%{
for t = 1:t_max
    figure(1)
    grid on
    hold on
    for cubes = 1:10
        plot_box_cage(final{cubes,t},all_points_each_COM);
    end
    hold off
end
%}
```

Crossover

```
function [new_population] = crossover(population, rank,mut)
new_population = population;
child1 = zeros(1,81);
child2 = zeros(1,81);
child1_part = zeros(1,30);
child2_part = zeros(1,30);
for i = 1:10
    x = randi([1,20],2);
    y = randi([1,50],1);
    child1 = population(x(1,1),:);
    child2 = population(x(1,2),:);
    child1_part(1,:) = child1(1,y:y+29);
    child2_part(1,:) = child2(1,y:y+29);
    child1(1,y:y+29) = child2_part(1,:);
    child2(1,y:y+29) = child1_part(1,:);
    a = randi([1,3],1);
    b = randi([1,27],1);
```

```
    if a == 1
        c=((3*(b-1))+1);
        child1(1,c) = (5000*rand);
        child2(1,c) = (5000*rand);
    end
    if a==2
        c=((3*(b-1))+2);
        child1(1,c) = -0.6 + 1.2*rand;
        child2(1,c) = -0.6 + 1.2*rand;
    end
    if a==3
        c= ((3*(b-1))+3);
        child1(1,c) = -pi +((2*pi)*rand);
        child2(1,c) = -pi +((2*pi)*rand);
    end

    new_population(2*i,:) = child2(1,:);
    new_population((2*i)-1,:) = child1(1,:);
end
for i = 1:90
    x = randi([1,200],2);
    y = randi([1,50],1);
    child1 = population(x(1,1),:);
    child2 = population(x(1,2),:);
    child1_part(1,:) = child1(1,y:y+29);
    child2_part(1,:) = child2(1,y:y+29);
    child1(1,y:y+29) = child2_part(1,:);
    child2(1,y:y+29) = child1_part(1,:);
    a = randi([1,3],1);
    b = randi([1,27],1);
    for j = 1:mut
        if a == 1
            c=((3*(b-1))+1);
            child1(1,c) = (5000*rand);
            child2(1,c) = (5000*rand);
        end
        if a==2
            c=((3*(b-1))+2);
            child1(1,c) = -0.6 + 1.2*rand;
            child2(1,c) = -0.6 + 1.2*rand;
        end
        if a==3
            c= ((3*(b-1))+3);
            child1(1,c) = -pi +((2*pi)*rand);
            child2(1,c) = -pi +((2*pi)*rand);
        end
    end
    new_population((2*i)+20,:) = child2(1,:);
    new_population((2*i)+19,:) = child1(1,:);
end
end
```

Evolution Trial

```
clc
clear all
%CREATING COM & points for each cube of ALL CUBES
cube_size1 = 3;
length = cube_size1;
width = cube_size1;
```

```
height = cube_size1;
COM = cell(length,width,height);
COM_location = zeros(3);
all_points_each_COM = cell(length,width,height);
points_location = zeros(1,3);
initial_height = 0;
for k = 1:height
    for j = 1:width
        for i = 1:length
            COM_location = [(0.5 + (i-1)),(0.5 + (j-1)),(0.5 + (k-1))];
            COM{i,j,k} = COM_location;
            pos = 1;
            for z=0:1
                for y = 0:1
                    for x= 0:1
                        points_location(1,pos)= (COM_location(1)+0.5+x);
                        points_location(2,pos)= (COM_location(2)+0.5+y);
                        points_location(3,pos)= (COM_location(3)+0.5+z);
                        pos = pos+1;
                    end
                end
            end
            all_points_each_COM{i,j,k} = points_location;
        end
    end
end
all_points_each_COM{3,3,3};
%COM CREATED
%CREATING POINTS LOCATION FOR EACH CUBE:
acceleration = cell(length+1, width+1, height+1);
vel = cell(length+1, width+1, height+1);
ini_location = cell(length+1,width+1,height+1);
point_location = zeros(3,1);
for k = 0:height
    for j = 0:width
        for i =0:length
            point_location(:,1) = [i;j;k+initial_height];
            ini_location{i+1,j+1,k+1} = point_location;
        end
    end
end
updated_locations = ini_location;
%Index and location for each point created. This includes the location and
%updated location of each point
ideal_dis = cell(length,width,height);
distance = zeros(8,8);
to_calc=zeros(3,8);
for k=1:height
    for j = 1:width
        for i = 1:length
            for row = 1:8
                l = all_points_each_COM{i,j,k}(1,row);
                m = all_points_each_COM{i,j,k}(2,row);
                n = all_points_each_COM{i,j,k}(3,row);
                %to_calc(:,row)=updated_locations{l,m,n};
                to_calc(1,row)=updated_locations{l,m,n}(1);
                to_calc(2,row)=updated_locations{l,m,n}(2);
                to_calc(3,row)=updated_locations{l,m,n}(3);
            end
```

```
            location = to_calc;
            for q=1:8
                for r = 1:q
                    distance(r,q) = sqrt(((location(1,q)-location(1,r))^2)+((location(2,q)-
location(2,r))^2)+((location(3,q)-location(3,r))^2));
                end
            end
            distance = triu(distance)+triu(distance,1)';
            ideal_dis{i,j,k} = distance;
        end
    end
end
ideals = zeros(8,8);
ideals = ideal_dis{1,1,1};
k_ground=10000;

%IDEAL_DISTANCES of all CENTRE of Masses has been calculated

%To calculate the acceleration of all points:
updated_locations = ini_location;
to_calc = zeros(3,8);
ideals = zeros(8,8);
acc = zeros(3,8);

for r = 1:height+1
    for q = 1: width+1
        for p = 1:length+1
            acceleration{p,q,r} = [0;0;0];
            vel{p,q,r} = [0;0;0];
        end
    end
end

size1 = 200;
segmenting =1
itermax = 20000;
plot_diversity = zeros(round(itermax/(size1*10)) , 1);
plot_dot_plot= zeros(itermax,1);
plot_learning = zeros(round(itermax/size1),1);
mut=1;
%%%%INITIALIZING DONE
while segmenting < 2
    k = zeros(1,81);
    size1 = 200;
    population = zeros(size1,81);
    random_k = zeros(size1,27);
    random_k = 0 + (5000*rand(size1,27));
    random_b = zeros(size1,27);
    random_b = -0.6 + 1.2*rand(size1,27);
    random_c = zeros(size1,27);
    random_c = -pi +((2*pi)*rand(size1,27));
    for i = 1:27
        population(:,1+3*(i-1)) =  random_k(:,i);
        population(:,2+3*(i-1)) =  random_b(:,i);
        population(:,3+3*(i-1)) =  random_c(:,i);
    end
    rank=zeros(size1,2);
    rank2 = zeros(size1,2);
    rank_global = zeros(size1*2,2);
```

```
    population2 = zeros(size1,81);
    pop_global = zeros(size1*2,81);
    pop_all = zeros(size1*2,83);
    compo = zeros(size1,1);
    travel = zeros(size1,1);

    parfor i=1:size1
        k = population (i,:);
        compo(i) = i;
        travel(i) = get_travel(to_calc,
all_points_each_COM,ideal_dis,updated_locations,ideals,acceleration,vel,k);
        plot_dot_plot(i,1) = travel(i);
    end

    rank(:,1)=compo;
    rank(:,2) =travel;
    pop_globe_all= zeros(size1*2,83);
    pop_globe_all(1:200,1:81) = population(:,:);
    pop_globe_all(1:200,82:83)= rank(:,:);
    %pop_globe_all = sortrows(pop_globe_all,83,'descend');
    pop_globe_all(:,82) = (1:400);
    population(:,:) = pop_globe_all(1:200,1:81);
    rank(:,:) = pop_globe_all(1:200,82:83);

    %Measurement of Population done
    % Lets try crossover
    gen=1;
    gen_count=0;
    gen_cunt = 0;
    iter=size1;
    while iter<itermax
        tic
        population2 = crossover(population,rank,mut);
        parfor i=1:size1
            k = population2(i,:);
            compo(i) = i;
            travel(i) = get_travel(to_calc,
all_points_each_COM,ideal_dis,updated_locations,ideals,acceleration,vel,k);
            plot_dot_plot(iter+i) = travel(i);
        end
        rank2(:,1)=compo;
        rank2(:,2) =travel;
        rank2(:,1)=rank2(:,1)+200;
        pop_globe_all(201:400,1:81) = population2(:,:);
        pop_globe_all(201:400,82:83) = rank2(:,:);
        pop_globe_all = sortrows(pop_globe_all,83,'descend');
        pop_globe_all(:,82) = (1:400);
        population(:,:) = pop_globe_all(1:200,1:81);
        rank(:,:) = pop_globe_all(1:200,82:83);
        iter = iter + size1
        if gen == 10
            gen_count = gen_count+1;
            [bre, wid] = size(unique(population));
            plot_diversity(gen_count) = bre*wid;
            if bre*wid<200
                mut = 3;
            end
            if bre*wid>300
                mut = 1;
```

```
            end
            gen=0;
        end
        gen=gen+1;
        gen_cunt = gen_cunt+1;
        plot_learning(gen_cunt) = rank(1,2);
        toc
    end
    segmenting = segmenting+1;
    if segmenting == 1
        pop1 = pop_globe_all(1:200,:);
    end
    if segmenting == 2
        pop2 = pop_globe_all(1:200,:);
    end
end
clear length
figure(1)
scatter([1:length(plot_dot_plot)],plot_dot_plot);
figure(2)
plot([1:length(plot_diversity)],plot_diversity);
figure(3)
plot([1:length(plot_learning)],plot_learning);
%{
pop_globe_all(1:50,:) = pop1(1:50,:);
pop_globe_all(51:100,:) = pop2(1:50,:);
pop_globe_all(101:150,:) = pop1(51:100,:);
pop_globe_all(151:200,:) = pop2(51:100,:);
pop_globe_all(:,82) = (1:400);
population(:,:) = pop_globe_all(1:200,1:81);
rank(:,:) = pop_globe_all(1:200,82:83);
    %Measurement of Population done
    % Lets try crossover
iter=200;
while iter<10000
    tic
    population2 = crossover(population,rank);
    parfor i=1:size1
        k = population2(i,:);
        compo(i) = i;
        travel(i) = get_travel(to_calc,
all_points_each_COM,ideal_dis,updated_locations,ideals,acceleration,vel,k);
    end
    rank2(:,1)=compo;
    rank2(:,2) =travel;
    rank2(:,1)=rank2(:,1)+200;
    pop_globe_all(201:400,1:81) = population2(:,:);
    pop_globe_all(201:400,82:83) = rank2(:,:);
    pop_globe_all = sortrows(pop_globe_all,83,'descend');
    pop_globe_all(:,82) = (1:400);
    population(:,:) = pop_globe_all(1:200,1:81);
    rank(:,:) = pop_globe_all(1:200,82:83);
    iter = iter + 200
    toc
end
%}
```

Evolution Trial 2

```
clc
clear all
%CREATING COM & points for each cube of ALL CUBES
cube_size = 3;
length = cube_size;
width = cube_size;
height = cube_size;
COM = cell(length,width,height);
COM_location = zeros(3);
all_points_each_COM = cell(length,width,height);
points_location = zeros(1,3);
initial_height = 0;
for k = 1:height
    for j = 1:width
        for i = 1:length
            COM_location = [(0.5 + (i-1)),(0.5 + (j-1)),(0.5 + (k-1))];
            COM{i,j,k} = COM_location;
            pos = 1;
            for z=0:1
                for y = 0:1
                    for x= 0:1
                        points_location(1,pos)= (COM_location(1)+0.5+x);
                        points_location(2,pos)= (COM_location(2)+0.5+y);
                        points_location(3,pos)= (COM_location(3)+0.5+z);
                        pos = pos+1;
                    end
                end
            end
            all_points_each_COM{i,j,k} = points_location;
        end
    end
end
all_points_each_COM{3,3,3};
%COM CREATED
%CREATING POINTS LOCATION FOR EACH CUBE:
acceleration = cell(length+1, width+1, height+1);
vel = cell(length+1, width+1, height+1);
ini_location = cell(length+1,width+1,height+1);
point_location = zeros(3,1);
for k = 0:height
    for j = 0:width
        for i =0:length
            point_location(:,1) = [i;j;k+initial_height];
            ini_location{i+1,j+1,k+1} = point_location;
        end
    end
end
updated_locations = ini_location;
%Index and location for each point created. This includes the location and
%updated location of each point
ideal_dis = cell(length,width,height);
distance = zeros(8,8);
to_calc=zeros(3,8);
for k=1:height
    for j = 1:width
        for i = 1:length
            for row = 1:8
                l = all_points_each_COM{i,j,k}(1,row);
                m = all_points_each_COM{i,j,k}(2,row);
```

```
                n = all_points_each_COM{i,j,k}(3,row);
                %to_calc(:,row)=updated_locations{l,m,n};
                to_calc(1,row)=updated_locations{l,m,n}(1);
                to_calc(2,row)=updated_locations{l,m,n}(2);
                to_calc(3,row)=updated_locations{l,m,n}(3);
            end
            location = to_calc;
            for q=1:8
                for r = 1:q
                    distance(r,q) = sqrt(((location(1,q)-location(1,r))^2)+((location(2,q)-
location(2,r))^2)+((location(3,q)-location(3,r))^2));
                end
            end
            distance = triu(distance)+triu(distance,1)';
            ideal_dis{i,j,k} = distance;
        end
    end
end
ideals = zeros(8,8);
ideals = ideal_dis{1,1,1};
k_ground=10000;

%IDEAL_DISTANCES of all CENTRE of Masses has been calculated

%To calculate the acceleration of all points:
updated_locations = ini_location;
to_calc = zeros(3,8);
ideals = zeros(8,8);
acc = zeros(3,8);

for r = 1:height+1
    for q = 1: width+1
        for p = 1:length+1
            acceleration{p,q,r} = [0;0;0];
            vel{p,q,r} = [0;0;0];
        end
    end
end


segmenting =1

%%%%INITIALIZING DONE
while segmenting < 3
    k = zeros(1,81);
    size = 200;
    population = zeros(size,81);
    random_k = zeros(size,27);
    random_k = 0 + (5000*rand(size,27));
    random_b = zeros(size,27);
    random_b = -0.6 + 1.2*rand(size,27);
    random_c = zeros(size,27);
    random_c = -pi +((2*pi)*rand(size,27));
    for i = 1:27
        population(:,1+3*(i-1)) =  random_k(:,i);
        population(:,2+3*(i-1)) =  random_b(:,i);
        population(:,3+3*(i-1)) =  random_c(:,i);
    end
    rank=zeros(200,2);
```

```
    rank2 = zeros(200,2);
    rank_global = zeros(400,2);
    population2 = zeros(200,81);
    pop_global = zeros(400,81);
    pop_all = zeros(400,83);
    compo = zeros(200,1);
    travel = zeros(200,1);
    parfor i=1:size
        k = population (i,:);
        compo(i) = i;
        travel(i) = get_travel(to_calc,
all_points_each_COM,ideal_dis,updated_locations,ideals,acceleration,vel,k);
    end
    rank(:,1)=compo;
    rank(:,2) =travel;
    pop_globe_all= zeros(400,83);
    pop_globe_all(1:200,1:81) = population(:,:);
    pop_globe_all(1:200,82:83)= rank(:,:)
    %pop_globe_all = sortrows(pop_globe_all,83,'descend');
    pop_globe_all(:,82) = (1:400);
    population(:,:) = pop_globe_all(1:200,1:81);
    rank(:,:) = pop_globe_all(1:200,82:83);
    %Measurement of Population done
    % Lets try crossover
    iter=200;
    while iter<500000
        tic
        population2 = crossover(population,rank);
        parfor i=1:size
            k = population2(i,:);
            compo(i) = i;
            travel(i) = get_travel(to_calc,
all_points_each_COM,ideal_dis,updated_locations,ideals,acceleration,vel,k);
        end
        rank2(:,1)=compo;
        rank2(:,2) =travel;
        rank2(:,1)=rank2(:,1)+200;
        pop_globe_all(201:400,1:81) = population2(:,:);
        pop_globe_all(201:400,82:83) = rank2(:,:);
        pop_globe_all = sortrows(pop_globe_all,83,'descend');
        pop_globe_all(:,82) = (1:400);
        population(:,:) = pop_globe_all(1:200,1:81);
        rank(:,:) = pop_globe_all(1:200,82:83);
        iter = iter + 200
        toc
    end
    segmenting = segmenting+1;
    if segmenting == 1
        pop1 = pop_globe_all(1:200,:);
    end
    if segmenting == 2
        pop2 = pop_globe_all(1:200,:);
    end
end
pop_globe_all(1:50,:) = pop1(1:50,:);
pop_globe_all(51:100,:) = pop2(1:50,:);
pop_globe_all(101:150,:) = pop1(51:100,:);
pop_globe_all(151:200,:) = pop2(51:100,:);
pop_globe_all(:,82) = (1:400);
```

```
population(:,:) = pop_globe_all(1:200,1:81);
rank(:,:) = pop_globe_all(1:200,82:83);
    %Measurement of Population done
    % Lets try crossover
iter=200;
while iter<10000
    tic
    population2 = crossover(population,rank);
    parfor i=1:size
        k = population2(i,:);
        compo(i) = i;
        travel(i) = get_travel(to_calc,
all_points_each_COM,ideal_dis,updated_locations,ideals,acceleration,vel,k);
    end
    rank2(:,1)=compo;
    rank2(:,2) =travel;
    rank2(:,1)=rank2(:,1)+200;
    pop_globe_all(201:400,1:81) = population2(:,:);
    pop_globe_all(201:400,82:83) = rank2(:,:);
    pop_globe_all = sortrows(pop_globe_all,83,'descend');
    pop_globe_all(:,82) = (1:400);
    population(:,:) = pop_globe_all(1:200,1:81);
    rank(:,:) = pop_globe_all(1:200,82:83);
    iter = iter + 200
    toc
end
```

Acceleration

```
function[acc] = get_acc2(location,ideal_dist,parameter,k_spring)
ideal_dist = ideal_dist*parameter;
distance = zeros(8,8);
k=k_spring;
m=1;
acc = zeros(3,8);
for i=1:8
    for j = 1:i
        distance(j,i) = sqrt(((location(1,i)-location(1,j))^2)+((location(2,i)-
location(2,j))^2)+((location(3,i)-location(3,j))^2));
    end
end
dist = triu(distance)+triu(distance,1)';
for i=1:8
    xfor = 0;
    yfor = 0;
    zfor = 0;
    for j=1:8
        if dist(i,j)>0
            xfor =((ideal_dist(i,j)-dist(i,j))*((location(1,i)-location(1,j))/dist(i,j))*(k))+
xfor;
            yfor =((ideal_dist(i,j)-dist(i,j))*((location(2,i)-location(2,j))/dist(i,j))*(k))+
yfor;
            zfor =((ideal_dist(i,j)-dist(i,j))*((location(3,i)-location(3,j))/dist(i,j))*(k))+
zfor;
        end
    end
    xacc = round((xfor*(1/m)),2);
    yacc = round((yfor*(1/m)),2);
    zacc = round((zfor*(1/m)),2) ;
    acc(1,i) = xacc;
```

```
    acc(2,i) = yacc;
    acc(3,i) = zacc;
end
```

Get Final Point

```
function [final] = get_final_points(to_calc,
all_points_each_COM,ideal_dis,updated_locations,ideals,acceleration,vel,params,final,cube)
time=3;
k_ground = 10000;
height = 3;
width =3;
length =3;

frict=0.7;
xfor =0;
yfor =0;
zfor =0;
g=10;
w= 10;
test_dist =10;
parameter = 1;
run_complete = 0;
dt = 0.005;
runs=0;
t=0;
updated_locations{1,1,1};
screenshot = 1;
time_inst =1;
total_mega_cubes=10;
for r = 1:height+1
    for q = 1: width+1
        for p = 1:length+1
            updated_locations{p,q,r} = updated_locations{p,q,r} + [0;5;5];
        end
    end
end
plot_box_cage(updated_locations,all_points_each_COM);
pause(2)
while run_complete == 0

    for z = 1:height
        for y = 1:width
            for x = 1:length
                for i = 1:8
                    l = all_points_each_COM{x,y,z}(1,i);
                    m = all_points_each_COM{x,y,z}(2,i);
                    n = all_points_each_COM{x,y,z}(3,i);
                    to_calc(:,i)=updated_locations{l,m,n};
                end
                parameter = 1 + params(x+((y-1)*3)+((z-1)*9)+1)*sin(w*t + params(x+((y-1)*3)+((z-
1)*9)+2));
                ideals = ideal_dis{x,y,z};
                k_spring = params(x+((y-1)*3)+((z-1)*9));
                acc = get_acc2(to_calc,ideals,parameter,k_spring);
                for i=1:8
                    l = all_points_each_COM{x,y,z}(1,i);
                    m = all_points_each_COM{x,y,z}(2,i);
                    n = all_points_each_COM{x,y,z}(3,i);
                    acceleration{l,m,n} = acceleration{l,m,n} + [acc(1,i);acc(2,i);acc(3,i)];
```

```
                end
            end
        end
    end
    for r = 1:height+1
        for q = 1: width+1
            for p = 1:length+1
                acceleration{p,q,r} = acceleration{p,q,r}+[xfor;yfor;zfor-g];
                vel{p,q,r} = (vel{p,q,r} + (acceleration{p,q,r}*dt))*1;
                updated_locations{p,q,r} = updated_locations{p,q,r} + (vel{p,q,r}*dt);
                run_complete;
                acceleration{p,q,r} = [0;0;0];
                if updated_locations{p,q,r}(3)<0

zacc=(k_ground*abs((updated_locations{p,q,r}(3))*(updated_locations{p,q,r}(3))));
                    acceleration{p,q,r} = [0;0;zacc];


                    frict_vel =sqrt((vel{p,q,r}(1)^2)+(vel{p,q,r}(2)^2));
                    if frict_vel>0
                        acceleration{p,q,r}(1) = -1*(vel{p,q,r}(1)/frict_vel)*frict*zacc;
                        acceleration{p,q,r}(2) = -1*(vel{p,q,r}(2)/frict_vel)*frict*zacc;
                    end
                end
            end
        end
    end
    t=t+dt;
    if t > time
        run_complete = 1;
    end
    runs = runs+1;
    if runs > 50
        runs = 0;
        plot_box_cage2(updated_locations,all_points_each_COM);
        final{cube,time_inst} = updated_locations;
        time_inst = time_inst +1;
        updated_locations{2,2,2};
        size(updated_locations);
        acceleration{1,1,1}(3)
        acceleration{1,1,4}(3)
        t
    end
end
run_complete
t
end
```

Get Travel

```
function [distance_travelled] = get_travel(to_calc,
all_points_each_COM,ideal_dis,updated_locations,ideals,acceleration,vel,params)
time=1;
k_ground = 10000;
height = 3;
width =3;
length =3;

frict=0.7;
xfor =0;
```

```
yfor =0;
zfor =0;
g=10;
w= 10;
test_dist =10;
parameter = 1;
run_complete = 0;
dt = 0.01;
runs=0;
t=0;
while run_complete == 0
    for z = 1:height
        for y = 1:width
            for x = 1:length
                for i = 1:8
                    l = all_points_each_COM{x,y,z}(1,i);
                    m = all_points_each_COM{x,y,z}(2,i);
                    n = all_points_each_COM{x,y,z}(3,i);
                    to_calc(:,i)=updated_locations{l,m,n};
                end
                parameter = 1 + params(3*(x+((y-1)*3)+((z-1)*9))-2+1)*sin(w*t + params(3*(x+((y-
1)*3)+((z-1)*9))));
                ideals = ideal_dis{x,y,z};
                k_spring = params(3*(x+((y-1)*3)+((z-1)*9))-2)/10;
                acc = get_acc2(to_calc,ideals,parameter,k_spring);
                for i=1:8
                    l=  all_points_each_COM{x,y,z}(1,i);
                    m = all_points_each_COM{x,y,z}(2,i);
                    n = all_points_each_COM{x,y,z}(3,i);
                    acceleration{l,m,n} = acceleration{l,m,n} + [acc(1,i);acc(2,i);acc(3,i)];
                end
            end
        end
    end
    for r = 1:height+1
        for q = 1: width+1
            for p = 1:length+1
                acceleration{p,q,r} = acceleration{p,q,r}+[xfor;yfor;zfor-g];
                vel{p,q,r} = (vel{p,q,r} + (acceleration{p,q,r}*dt))*1;
                updated_locations{p,q,r} = updated_locations{p,q,r} + (vel{p,q,r}*dt);
                run_complete;
                if updated_locations{p,q,r}(1)> test_dist
                    run_complete = 1;
                end
                acceleration{p,q,r} = [0;0;0];
                if updated_locations{p,q,r}(3)<0

zacc=(k_ground*abs((updated_locations{p,q,r}(3))*(updated_locations{p,q,r}(3))));
                    acceleration{p,q,r} = [0;0;zacc];
                    frict_vel =sqrt((vel{p,q,r}(1)^2)+(vel{p,q,r}(2)^2));
                    if frict_vel>0
                        acceleration{p,q,r}(1) = -1*(vel{p,q,r}(1)/frict_vel)*frict*zacc;
                        acceleration{p,q,r}(2) = -1*(vel{p,q,r}(2)/frict_vel)*frict*zacc;
                    end
                end
            end
        end
    end
```

```
    t=t+dt;
    if t > time
        run_complete = 1;
    end
    runs = runs+1;
    %REMOVE '%'  on LINES UNDER this to print
    %if runs > 200
    %    runs = 0;
    %    plot_box_cage(updated_locations,all_points_each_COM);
    %end
end
distance_travelled = updated_locations{2,2,2}(1);
end
```

Plot Box Cage

```
function plot_box_cage2(points,COM_points)
location = zeros(3,8);
v= zeros(2,3);
for i = 1:8
    l = COM_points{1,1,1}(1,i);
    m = COM_points{1,1,1}(2,i);
    n = COM_points{1,1,1}(3,i);
    location(:,i)=points{l,m,n};
end
figure(1)
v(1,:)=[location(1,1),location(2,1),location(3,1)];
v(2,:)=[location(1,2),location(2,2),location(3,2)];
plot3(v(:,1),v(:,2),v(:,3),'r');
hold on
grid on;

%{
for z=1:3
    for y=1:3
        for x = 1:3
            for i = 1:8
                l = COM_points{x,y,z}(1,i);
                m = COM_points{x,y,z}(2,i);
                n = COM_points{x,y,z}(3,i);
                location(:,i)=points{l,m,n};
            end
            for i=1:8
                %scatter3(location(1,i),location(2,i),location(3,i),'filled');
                v(1,:) = [location(1,i),location(2,i),location(3,i)];
                for j=1:i
                    v(2,:)= [location(1,j),location(2,j),location(3,j)];
                    plot3(v(:,1),v(:,2),v(:,3),'r');
                end
            end
        end
    end
end
%}
for z=1:4
    for y=1:4
        for x = 1:4
            location(:,i)=points{x,y,z};
            scatter3(location(1,i),location(2,i),location(3,i),'filled');
        end
```

```
    end
end

axis([-1 10 -1 10 -1 10]);
%axis([location(1,1)-2 location(1,1)+2 location(2,1)-2 location(2,1)+2 location(3,1)-2
location(3,1)+2])
%surface(location(1,:),location(2,:),location(3,:));
hold off
end
```