Project (Phase C)

Nattapong Ounanong

UNI: no2313

Course: MECS 4510 – Evolutionary Computation

Instructor: Hod Lipson

Date Submitted: December 21, 2018

Grace Hours: 63 used / 4 remaining

Results

Table 1: Parameters of the fastest robot

Material	K [N/m]	a	b	c
Soft Muscle	1000	0.1	0.5	Pi
Bone	10000	0.1	0	0
Hard Muscle	5000	0.1	0.1	Pi
Air	0	0.2	0	0

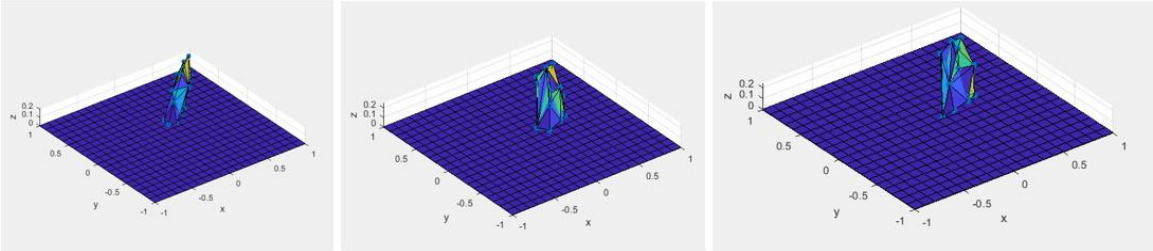

Figure 1: Fastest robot in three frames of its motion

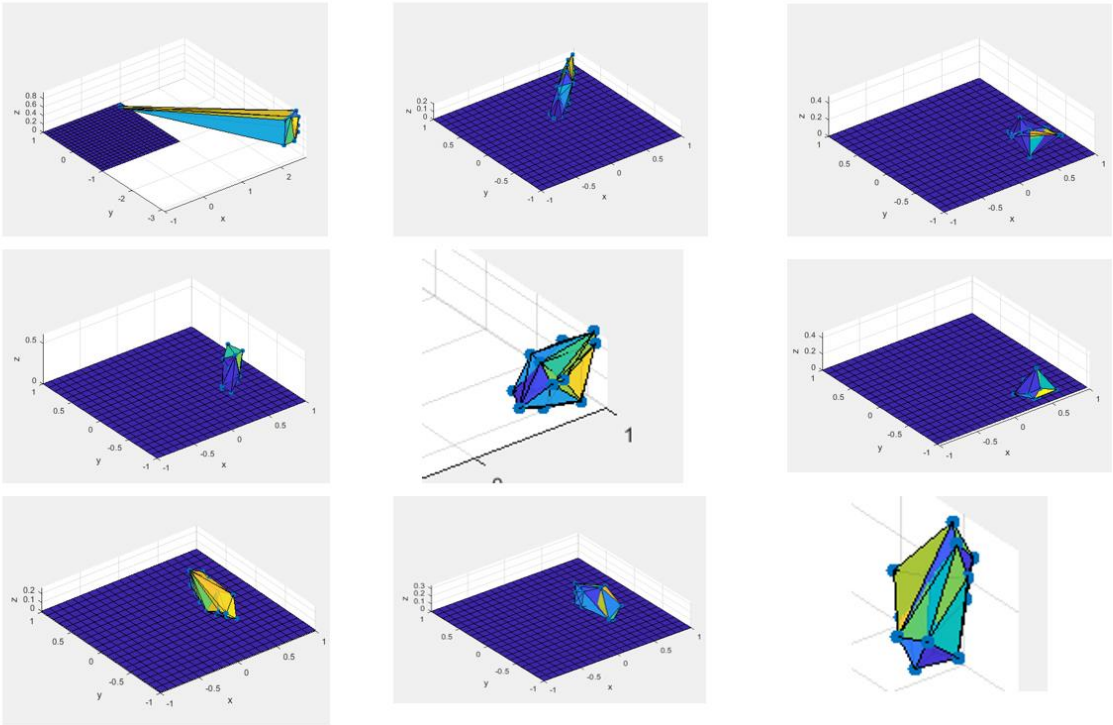

Figure 2: Robot zoo

Video on Youtube consist of tetrahedron robot falling and bounce on the ground and the video of the robot trying to improve its locomotion.

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

Method

The simulation of the bouncing robot calculated the motion of the robot for 5 seconds and the simulation has begun when the robot is rest on the ground. The code evaluated the robot position with the time increment of 0.0001s and the value of damping coefficient is 0.999; however, it is found later that the damping coefficient is too low so that it prevented the robot from moving and was later change into 0.9999. In order to make the robot move, the friction force is included, in which the chosen friction coefficient is 0.8, into the simulation and it would let the rest length of the spring change over time by using sine function as $L_o = a + bsin(\omega t+c)$, where ω is 10. To visualize the result of the simulation and prevent slow motion caused from plotting too frequently, the simulation would only plot the robot in every 1000 iterations. For the representation of the robot, due to computational speed of MATLAB, the tetrahedron is chosen as an original shape in evolving morphologythe of the robot. When it was created, additional masses were added to the robot to create another tetrahedron and connect ot the original. This result in the robot composed of several tetrahedrons.

In order to make the robot move, four spring parameters was predefined as shown in **Table 1** which would be assigned to each spring of the robot. The robot locomotion is evolving by assign the number from 1-4 which represent to specific material in the order of the spring index. For example, spring number 1 to 10 have number of 1 which would become soft muscle of the robot and these numbers would be in sequence with different range from 1-4. Initially, the range of material index was randomly generated and assigned to springs with the population of 30. To perform crossover, the sequence of the material index of the robot was exchanged and selection method used in this assignment is priority encoding which would rank the robot base on fitness, then replace the selected worst 50% by the child of the best 50%. Mutation is performed by twisting the end point of the spring, i.e. bit string where the value change from 1 to 2, which would change the spring material to maintain diversity. In the first attempt, the rule is set so that mutation would be performed repeatedly for 10 iterations and have 50% chance of performing this variation operator. However, the result shown that the population had become identical rather quickly and the algorithm was changed by allow 100% chance of the population to undergo mutation. This method does not help much in term of diversity maintenance as the newly generated population was discarded in the next generation. As of using this representation and plotting method, if the spring material is air, it will not contribute movement to the robot and not be plotted.

After 100 generation, the diversity decline which can be detected from the dot plot and data generated from running the code. Some robots are identical so that the fitness is identical and can no longer be seen from dot plot. This representation does not work very well as the number of populations is to low and robot size is small which only contain at most 69 springs and 25 masses. The diversity also decreased significantly after crossover operation. This might come from selection method as the child always carried gene from their parent which replace the parents that within the worst 50% and could be the reason for premature convergence as shown in **Figure 3**. Alternative mutation operation is to rearrange the order of material index within the robot to create diversity; however, due to time limit, thi idea wasn't implement into this assignment. The cycle of the sinusoidal activation is approximately 0.7 seconds and within that time the robot had travelled for 0.5146 meters in which the robot has maximum diameter of 1.1314 meters. Therefore, the speed of the robot is 45.48% diameter per cycle which equivalent to 0.8359 m/s. As the fitness is measured as the displacement of the center of mass of the robot from the origin. As a result, the total distance travelled of the robot cannot be measured as, from the video, the robot moved around the origin in circle which means that the actual distance travelled might be larger than estimated values.

Performance Plot

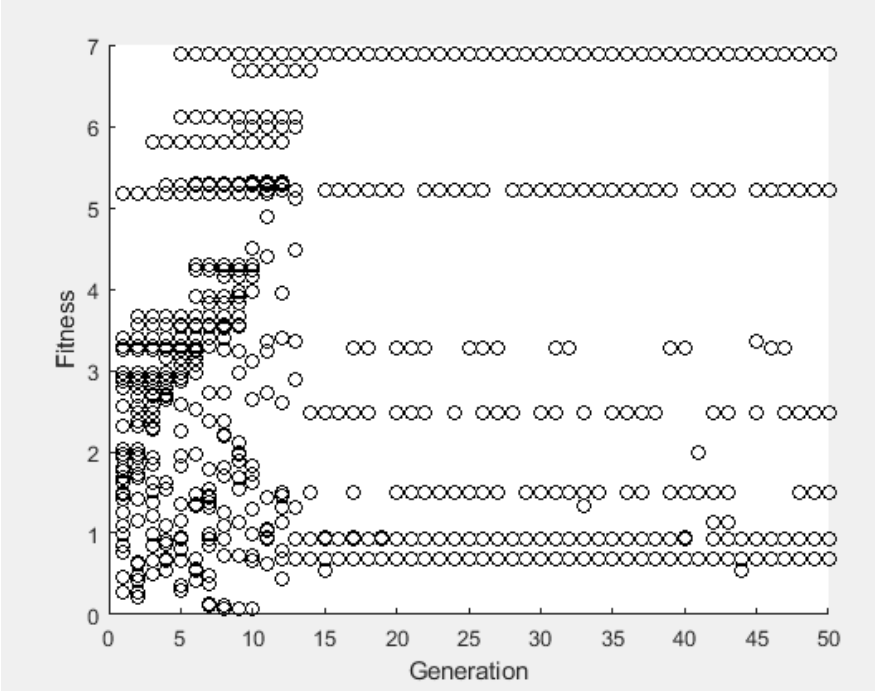

Figure 3: Dot Plot

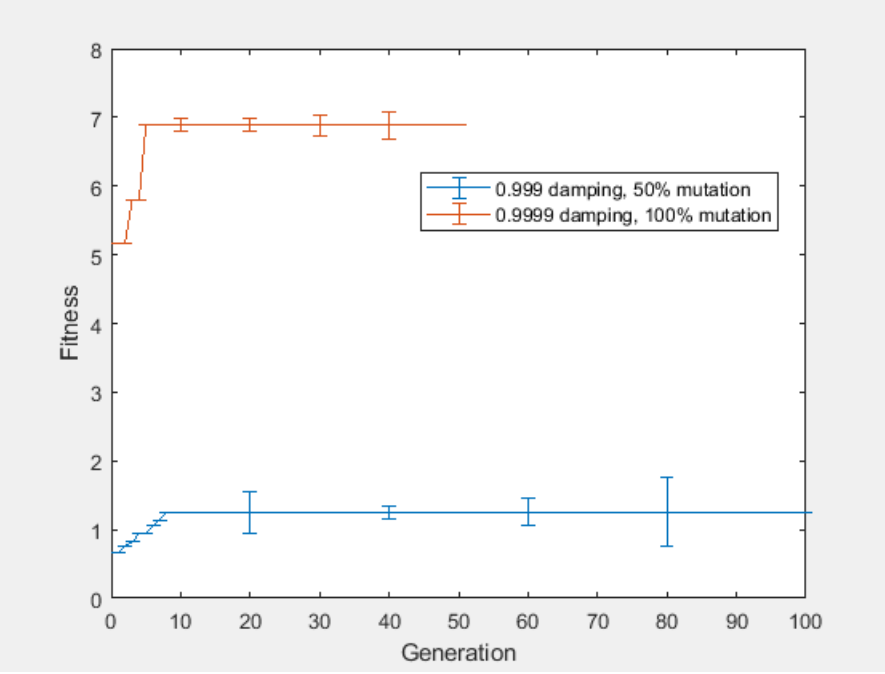

Figure 4: Learning plot

Appendix (Matlab Code)

```
clear;clc;close all;
% initiallized constant
initPos = 0.2; k = 1e4;g = [0,0,-9.81];dt=0.0001;z = 0;T = 0;iterate = 1;
 Fc = zeros(1,8);
% dt = 0.1;
%     create mass matrix in which tha the 1st column is mass of vertex in
%     kg, column 2-4 are 3D position vector, column 5-7 are velocity vector
%     and column 8-10 are acceleration vector
mass = tetrahedron();
% Create Spring array having spring constant in column one, 2nd column is
% rest length of spring and column 3 and 4 are mass that the spring
% connects to
% Loop start here

count  = 1;
% spring_store = spring;
spring_store = spring_gen(mass);
% sL = size(spring,1);
% These two variable might be useful later
mL = size(mass,1);
mg = mass(:,1).*g;

% ####################################################################
% test code
% [mass, spring] = appendMass(mass,spring);
% ####################################################################

% spring = [spring;newSpring];
% ####################################################################
% This portion woujld include only EA algorithm
T = gen_prop();
   population = 30; % Number of populations
   Eval = 50; % Number of generations
for pop = 1:population
    [Prop_Table, spInd] = Indirect_code(T,spring_store); %Indirect Encoding
    %        Store parameter in these variables
    spInd_gen([1:size(Prop_Table,1)],pop) = spInd;
    k_encode([1:size(Prop_Table,1)],pop) = Prop_Table(:,1);
    a([1:size(Prop_Table,1)],pop) = Prop_Table(:,2);
    b([1:size(Prop_Table,1)],pop) = Prop_Table(:,3);
    c([1:size(Prop_Table,1)],pop) = Prop_Table(:,4);
    material([1:size(Prop_Table,1)],pop) = Prop_Table(:,5);
end
for e = 1:Eval
    e
    for pop = 1:population
        Prop_Table = [k_encode(:,pop),a(:,pop),b(:,pop),c(:,pop)];
        spring_store2 = spring_store;
        for add_k = 1:size(spring_store,1)
            spring_store2(add_k,1) = Prop_Table(add_k,1);
        end
        dist(pop,1) = distance(Prop_Table,spInd_gen(:,pop), spring_store2, mass);
    end

%%     EA -- Crossover and mutation
%     Sort population
    [dist_sort, dist_ind] = sort(dist,'descend');
    k_encode = k_encode(:,dist_ind);
    a = a(:,dist_ind);
    b = b(:,dist_ind);
    c = c(:,dist_ind);
    material = material(:,dist_ind);
    if mod(e,10)==0
        Fastest = [k_encode(:,1), a(:,1), b(:,1), c(:,1)];
        spring_store2 = spring_store;
        for add_k = 1:size(spring_store,1)
            spring_store2(add_k,1) = Fastest(add_k,1);
        end
        Simulate_Fastest(Fastest(:,[1:4]),spInd_gen(:,1), spring_store2, mass, material(:,30));
```

```
        max_distance = dist_sort(1)
    end
    spInd_gen = spInd_gen(:,dist_ind);
    learning_plot(e,1) = dist_sort(1);
    dot_plot([1:population],e) = dist_sort;
    choose_half = floor(size(dist_sort,1)*0.5);
    if mod(choose_half,2)== 1
        choose_half = choose_half - 1;
    end
%     Perform crossover for 50% of the total population
%     k_encode
    for cross = 1:2:choose_half
        ranIndex = randi(size(spring_store,1));
        ranIndex = round(ranIndex/2);
        if mod(ranIndex,2) == 1
            ranIndex = ranIndex-1;
        end
        [k_encode(:,choose_half+cross), k_encode(:,choose_half+cross+1)] =
Cross_Spring(k_encode(:,cross), k_encode(:,cross+1),ranIndex);
        [a(:,choose_half+cross), a(:,choose_half+cross+1)] = Cross_Spring(a(:,cross),
a(:,cross+1),ranIndex);
        [b(:,choose_half+cross), b(:,choose_half+cross+1)] = Cross_Spring(b(:,cross),
b(:,cross+1),ranIndex);
        [c(:,choose_half+cross), c(:,choose_half+cross+1)] = Cross_Spring(c(:,cross),
c(:,cross+1),ranIndex);
        [material(:,choose_half+cross), material(:,choose_half+cross+1)] =
Cross_Spring(material(:,cross), material(:,cross+1),ranIndex);
    end
%     k_encode
%     return
for i = 1:30
    ransom_ind = randi([round(size(a,2)/2),size(a,2)]); %randomly select population
    random_sp =
[k_encode(:,ransom_ind),a(:,ransom_ind),b(:,ransom_ind),c(:,ransom_ind),material(:,ransom_ind)];
    random_sp = Mutation(random_sp);
    for j = 1:size(a,1)
        k_encode(j,ransom_ind) = T(random_sp(j,5),1);
        a(j,ransom_ind) = T(random_sp(j,5),2);
        b(j,ransom_ind) = T(random_sp(j,5),3);
        c(j,ransom_ind) = T(random_sp(j,5),4);
    end
    material(:,ransom_ind) = random_sp(:,5);
end
%     Perform Mutation of the worst 50% of the population
% ###########################################################################
% The following line of code would attempt to add additional mass to the
% robot
%     [mass, spring] = appendMass(mass,spring);
%     Distance_2 = distance_2(Prop_Table, spInd, spring, mass);
figure(1)
scatter(zeros(1,size(dot_plot,1))+e,dot_plot(:,e),'k')
hold on
xlabel('Generation');ylabel('Fitness')
figure(2)
plot([1:e],learning_plot)
xlabel('Generation');ylabel('Fitness')
end

figure(1)
for dot = 1:population
scatter([1:e],dot_plot(dot,:),'k')
hold on
end
xlabel('Generation');ylabel('Fitness')
figure(3)
plot([1:e],learning_plot)
xlabel('Generation');ylabel('Fitness')
save('data.mat')

function [] = drawing(iterate,x,y,z)
%UNTITLED2 Summary of this function goes here
```

```
%   Detailed explanation goes here
frame = 1;
if mod(iterate,100)==0
    DT = delaunayTriangulation(x,y,z);
    tetramesh(DT);
    hold on
%     axis ([-frame frame -frame frame 0 frame]);
    xlabel('x');ylabel('y');zlabel('z');
    grid on
    scatter3(x,y,z,'filled')
    [x, y] = meshgrid(-frame:0.1:frame); % Generate x and y data
    z = zeros(size(x, 1)); % Generate z data
    hold on
    surf(x, y, z) % Plot the surface
%     colormap([1 0.85 1])
    drawnow
    pause(0.0001)
    grid on
    hold off
end
end


function D = Simulate_Fastest(Prop_Table, spInd, spring, mass, material)
%Author: Nattapong Ounanong
T = 0;iterate = 1;
sL = size(spring,1);k = 1e4;
k = 1e4;g = [0,0,-9.81];dt=0.0001;z = 0;iterate = 1;
Fc = zeros(1,8);uk = 0.8;
mL = size(mass,1);sL = size(spring,1);
mg = mass(:,1).*g;
count = 1;
Lo = spring(:,2);
while T <= 1
%     Update current spring length
    count  = 1;
%     This is breathing function
% Lo = breathNow(Lo, Prop_Table,T, spInd);

for link = 1:size(spring,1)
    %         determine length of spring
    Lx(count) = (mass(spring(link,3),2) - mass(spring(link,4),2));
    Ly(count) = (mass(spring(link,3),3) - mass(spring(link,4),3));
    Lz(count) = (mass(spring(link,3),4) - mass(spring(link,4),4));
    L(count) = sqrt(Lx(count)^2+Ly(count)^2+Lz(count)^2);
    %             Unit vector would be use for calculating new orientation of
    %             the spring
    unitVector(count,:) = [Lx(count)/L(count),Ly(count)/L(count),Lz(count)/L(count)];
    count = count +1;
end

%     Calculate spring force
    x = []; y =[]; z=[];
    for F = 1:sL
        sF(spring(F,3),F) = spring(F,1)*(Lo(F)-L(F));

        sFx(spring(F,3),F) = sF(spring(F,3),F)*unitVector(F,1);
        sFx(spring(F,4),F) = -sFx(spring(F,3),F);
        sFy(spring(F,3),F) = sF(spring(F,3),F)*unitVector(F,2);
        sFy(spring(F,4),F) = -sFy(spring(F,3),F);
        sFz(spring(F,3),F) = sF(spring(F,3),F)*unitVector(F,3);
        sFz(spring(F,4),F) = -sFz(spring(F,3),F);
        if material(F) == 4
        else
            x = [x;mass(spring(F,[3:4]),2)];
            y = [y;mass(spring(F,[3:4]),3)];
            z = [z;mass(spring(F,[3:4]),4)];
        end
    end
    % Sum all spring force acting on each mass
    sFxm = sum(sFx,2);
```

```
    sFym = sum(sFy,2);
    sFzm = sum(sFz,2);
    % iterate though each mass

%         Sum reaction force
if sum(Fc) == 0
    FcmG = 0;
    zSum = zeros(1,size(mass,1));
    damping = 0.9999;
else
    proportionZ = sum(unitVector(:,3));
    for mG = 1:mL
        for Zforce = 1:size(sFz,2)
            FcmG(mG,Zforce) = Fc(mG) * unitVector(Zforce,3)/proportionZ;
        end
        zSum(mG) = sum(FcmG(mG,:));
    end
    damping = 0.9999;
end

    for mG = 1:mL
%         Caculate equivalent spring force
%         summation of forces
            sumFx = sFxm(mG)+mg(mG,1);
            sumFy = sFym(mG)+mg(mG,2);
            sumFz = sFzm(mG)+mg(mG,3)+zSum(mG);
        % Friction force: Assume rubber on concrete
        if mass(mG,4)<=0
            Fh = sqrt(sumFx^2+sumFy^2);
            Ff = sumFz*uk;
            if Fh > Ff
                if Fh == 0
                    Frictionx = 0;
                    Friction y = 0;
                else
                    Frictionx = sumFx/Fh*Ff;
                    Friction y = sumFy/Fh*Ff;
                end
                sumFx = sumFx+Frictionx;
                sumFy = sumFx+Friction y;
            else
                sumFx = 0;
                sumFy = 0;
            end
        else
            Fh = 0;
        end

        %         acceleration
        mass(mG,8) = sumFx/mass(mG,1);
        mass(mG,9) = sumFy/mass(mG,1);
        mass(mG,10) = sumFz/mass(mG,1);

        %         velocity
        mass(mG,5) = (mass(mG,5)+dt*mass(mG,8))*damping;
        mass(mG,6) = (mass(mG,6)+dt*mass(mG,9))*damping;
        mass(mG,7) = (mass(mG,7)+dt*mass(mG,10))*damping;

        %         position
        mass(mG,2) = mass(mG,2)+dt*mass(mG,5);
        mass(mG,3) = mass(mG,3)+dt*mass(mG,6);
%         mass(mG,2) = mass(mG,2);
%         mass(mG,3) = mass(mG,3);
        mass(mG,4) = mass(mG,4)+dt*mass(mG,7);
%         make sure the cube would not go below ground
%         Bounce the ball if individual mass is below ground level
        if mass(mG,4) < 0
            Fc(1,mG) = -k*mass(mG,4);
            Eg(mG,iterate) = k/2*mass(mG,4)^2;
        else
            Fc(1,mG) = 0;
```

```
            Eg(mG,iterate) = 0;
        end
    end
drawing(iterate,x,y,z)
T = T+dt;
iterate = iterate + 1;
end
% Determine the center of the cube
sumX = sum(mass(:,2))/size(mass,1);
sumY = sum(mass(:,3))/size(mass,1);
norm([sumX,sumY])
D = 0;
end
function [Prop_table] = Mutation(Prop_table)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
prob = rand;
% random_sp = randi(size(Prop_table,1));
% r2 = randi(size(Prop_table,1));

material = Prop_table(:,5);
Soft_Muscle = find(Prop_table(:,5) == 1);
Bone = find(Prop_table(:,5) == 2);
hard_Muscle = find(Prop_table(:,5) == 3);
Air = find(Prop_table(:,5) == 4);

if length(Bone) >= 1 && length(Soft_Muscle) >=1 && length(hard_Muscle) >=1 && length(Air) >= 1
    sEnd = Soft_Muscle(end);
    bEnd = Bone(end);
    hEnd = hard_Muscle(end);
    card = randi([1 2]);
        switch card
            case 1
                material(sEnd) = 2; %Change soft muscle to bone
                material(bEnd) = 3; %Change bone to hard muscle
            case 2
                material(sEnd+1) = 1; %Change bone to soft muscle
                material(bEnd+1) = 2; %Change hard muscle to bone
        end
end
Prop_table(:,5) = material;
end


function [Lo] = breathNow(Lo,Table,time, sp_index)
% This function
mod_spring = sp_index(sp_index~=0);
noSpring = length(mod_spring);
for k = 1:noSpring
Lo(mod_spring(k)) = Table(k,2)+Table(k,3)*sin(10*time+Table(k,4));
end
end


function [T] = gen_prop()
%Author: Nattapong Ounanong, Graduate student Columbia University
% Created: November 26, 2018
%   this code would generate random property of the spring, then output it
%   as table of spring property - a,b,c,k
%   the rest length of the spring would change by utilizing sinewave with
%   aforementioned parameter

%   a is initial length of the spring
%   b is amplitude of the sinewave
%   c is phase shift of the sinewave
%   k is spring coefficient
% for k = 1:100
%     T(k,1) = randi([1000 10000]); %This is 'k'
%     T(k,2) = rand*0.1; %This is 'a'
%     T(k,3) = rand*0.01; %This is 'b' - amplitude
%     T(k,4) = rand*pi*(rand-rand); %This is 'c' - phase shift
```

```
% end
    T = [];
    T(1,:) = [1000 0.1 0.5 pi 1]; % this is soft muscle
    T(2,:) = [10000 0.1 0 0 2]; % This is bone
    T(3,:) = [5000 0.1 0.1 pi 3]; % this is hard muscle
    T(4,:) = [0 0.2 0 0 4]; % this is air
end


function [mass] =  tetrahedron
% This function generate a tetrahedron shape
%   Detailed explanation goes here
initPos = 0;
mass = [0.1, 0.1 ,0.1 ,initPos , 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.2 ,0.1 ,initPos , 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.1 ,0.2 ,initPos , 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.2 ,0.2 ,initPos+0.2, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.2 ,0.3 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.3 ,0.2 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.3 ,0.3 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.3 ,0.4 ,initPos+0.2, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.4 ,0.3 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.4 ,0.4 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.4 ,0.5 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.5 ,0.4 ,initPos+0.2, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.5 ,0.5 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.5 ,0.6 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.6 ,0.5 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.6 ,0.6 ,initPos+0.2, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.6 ,0.7 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.7 ,0.6 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.7 ,0.7 ,initPos+0.2, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.7 ,0.8 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.8 ,0.7 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.8 ,0.8 ,initPos+0.2, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.8 ,0.9 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.9 ,0.8 ,initPos, 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.9 ,0.9 ,initPos+0.2, 0 ,0 ,0 ,0 ,0 ,0;];
end


function D = distance(Prop_Table, spInd, spring, mass)
%Author: Nattapong Ounanong
T = 0;iterate = 1;
sL = size(spring,1);k = 1e4;
k = 1e4;g = [0,0,-9.81];dt=0.0001;z = 0;iterate = 1;
Fc = zeros(1,8);uk = 0.8;
mL = size(mass,1);sL = size(spring,1);
mg = mass(:,1).*g;
count = 1;
Lo = spring(:,2);
while T <= 2
    %     Update current spring length
    count  = 1;
    %     This is breathing function
    % has a dimension of 1 by number of spring in the cube
    Lo = breathNow(Lo, Prop_Table, T, spInd);

    for link = 1:size(spring,1)
        %         determine length of spring
%         spring(link,3)
%         spring(link,4);
        Lx(count) = (mass(spring(link,3),2) - mass(spring(link,4),2));
        Ly(count) = (mass(spring(link,3),3) - mass(spring(link,4),3));
        Lz(count) = (mass(spring(link,3),4) - mass(spring(link,4),4));
        L(count) = sqrt(Lx(count)^2+Ly(count)^2+Lz(count)^2);
        %             Unit vector would be use for calculating new orientation of
        %             the spring
        unitVector(count,:) = [Lx(count)/L(count),Ly(count)/L(count),Lz(count)/L(count)];
        count = count +1;
    end
```

```
%     Calculate spring force
x = []; y =[]; z=[];
for F = 1:sL
    sF(spring(F,3),F) = spring(F,1)*(Lo(F)-L(F));

    sFx(spring(F,3),F) = sF(spring(F,3),F)*unitVector(F,1);
    sFx(spring(F,4),F) = -sFx(spring(F,3),F);
    sFy(spring(F,3),F) = sF(spring(F,3),F)*unitVector(F,2);
    sFy(spring(F,4),F) = -sFy(spring(F,3),F);
    sFz(spring(F,3),F) = sF(spring(F,3),F)*unitVector(F,3);
    sFz(spring(F,4),F) = -sFz(spring(F,3),F);

    x = [x;mass(spring(F,[3:4]),2)];
    y = [y;mass(spring(F,[3:4]),3)];
    z = [z;mass(spring(F,[3:4]),4)];

end

% Sum all spring force acting on each mass
sFxm = sum(sFx,2);
sFym = sum(sFy,2);
sFzm = sum(sFz,2);

% iterate though each mass

%         Sum reaction force
if sum(Fc) == 0
    FcmG = 0;
    zSum = zeros(1,size(mass,1));
    damping = 0.9999;
else
    proportionZ = sum(unitVector(:,3));
    for mG = 1:mL
        for Zforce = 1:size(sFz,2)
            FcmG(mG,Zforce) = Fc(mG) * unitVector(Zforce,3)/proportionZ;
        end
        zSum(mG) = sum(FcmG(mG,:));
    end
    damping = 0.9999;
end

for mG = 1:mL
    %         Caculate equivalent spring force
    %         summation of forces
    sumFx = sFxm(mG)+mg(mG,1);
    sumFy = sFym(mG)+mg(mG,2);
    sumFz = sFzm(mG)+mg(mG,3)+zSum(mG);

    % Friction force: Assume rubber on concrete
    if mass(mG,4)<=0
        Fh = sqrt(sumFx^2+sumFy^2);
        Ff = sumFz*uk;
        if Fh > Ff
            if Fh == 0
                Frictionx = 0;
                Frictiony = 0;
            else
                Frictionx = sumFx/Fh*Ff;
                Frictiony = sumFy/Fh*Ff;
            end
            sumFx = sumFx+Frictionx;
            sumFy = sumFx+Frictiony;
        else
            sumFx = 0;
            sumFy = 0;
        end
    else
        Fh = 0;
    end
    %         acceleration
```

```
        mass(mG,8) = sumFx/mass(mG,1);
        mass(mG,9) = sumFy/mass(mG,1);
        mass(mG,10) = sumFz/mass(mG,1);
        %         velocity
        mass(mG,5) = (mass(mG,5)+dt*mass(mG,8))*damping;
        mass(mG,6) = (mass(mG,6)+dt*mass(mG,9))*damping;
        mass(mG,7) = (mass(mG,7)+dt*mass(mG,10))*damping;
        %         position
        mass(mG,2) = mass(mG,2)+dt*mass(mG,5);
        mass(mG,3) = mass(mG,3)+dt*mass(mG,6);
        mass(mG,4) = mass(mG,4)+dt*mass(mG,7);

        %         make sure the cube would not go below ground
        %         Bounce the ball if individual mass is below ground level
        if mass(mG,4) < 0
            Fc(1,mG) = -k*mass(mG,4);
            Eg(mG,iterate) = k/2*mass(mG,4)^2;
        else
            Fc(1,mG) = 0;
            Eg(mG,iterate) = 0;
        end
    end
    T = T+dt;
    iterate = iterate + 1;
end
% Determine the center of the cube
sumX = sum(mass(:,2))/size(mass,1);
sumY = sum(mass(:,3))/size(mass,1);
D = norm([sumX,sumY]);
end


function [spring] = spring_gen(mass)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
k = 1e4;
spring = [k, sqrt((mass(2,2) - mass(1,2))^2+(mass(2,3) - mass(1,3))^2+(mass(2,4) - mass(1,4))^2),
1, 2;
          k, sqrt((mass(3,2) - mass(1,2))^2+(mass(3,3) - mass(1,3))^2+(mass(3,4) - mass(1,4))^2),
1, 3;
          k, sqrt((mass(4,2) - mass(1,2))^2+(mass(4,3) - mass(1,3))^2+(mass(4,4) - mass(1,4))^2),
1, 4;
          k, sqrt((mass(3,2) - mass(2,2))^2+(mass(3,3) - mass(2,3))^2+(mass(3,4) - mass(2,4))^2),
2, 3;
          k, sqrt((mass(4,2) - mass(2,2))^2+(mass(4,3) - mass(2,3))^2+(mass(4,4) - mass(2,4))^2),
2, 4;
          k, sqrt((mass(4,2) - mass(3,2))^2+(mass(4,3) - mass(3,3))^2+(mass(4,4) - mass(3,4))^2),
3, 4;
          k, sqrt((mass(2,2) - mass(5,2))^2+(mass(2,3) - mass(5,3))^2+(mass(2,4) - mass(5,4))^2),
5, 2;
          k, sqrt((mass(3,2) - mass(5,2))^2+(mass(3,3) - mass(5,3))^2+(mass(3,4) - mass(5,4))^2),
5, 3;
          k, sqrt((mass(4,2) - mass(5,2))^2+(mass(4,3) - mass(5,3))^2+(mass(4,4) - mass(5,4))^2),
5, 4;
          k, sqrt((mass(3,2) - mass(6,2))^2+(mass(3,3) - mass(6,3))^2+(mass(3,4) - mass(6,4))^2),
6, 3;
          k, sqrt((mass(4,2) - mass(6,2))^2+(mass(4,3) - mass(6,3))^2+(mass(4,4) - mass(6,4))^2),
6, 4;
          k, sqrt((mass(5,2) - mass(6,2))^2+(mass(5,3) - mass(6,3))^2+(mass(5,4) - mass(6,4))^2),
6, 5;
          k, sqrt((mass(4,2) - mass(7,2))^2+(mass(4,3) - mass(7,3))^2+(mass(4,4) - mass(7,4))^2),
7, 4;
          k, sqrt((mass(5,2) - mass(7,2))^2+(mass(5,3) - mass(7,3))^2+(mass(5,4) - mass(7,4))^2),
7, 5;
          k, sqrt((mass(6,2) - mass(7,2))^2+(mass(6,3) - mass(7,3))^2+(mass(6,4) - mass(7,4))^2),
7, 6;
          k, sqrt((mass(5,2) - mass(8,2))^2+(mass(5,3) - mass(8,3))^2+(mass(5,4) - mass(8,4))^2),
8, 5;
          k, sqrt((mass(6,2) - mass(8,2))^2+(mass(6,3) - mass(8,3))^2+(mass(6,4) - mass(8,4))^2),
8, 6;
```

```
          k, sqrt((mass(7,2) - mass(8,2))^2+(mass(7,3) - mass(8,3))^2+(mass(7,4) - mass(8,4))^2),
8, 7;
          k, sqrt((mass(6,2) - mass(9,2))^2+(mass(6,3) - mass(9,3))^2+(mass(6,4) - mass(9,4))^2),
9, 6;
          k, sqrt((mass(7,2) - mass(9,2))^2+(mass(7,3) - mass(9,3))^2+(mass(7,4) - mass(9,4))^2),
9, 7;
          k, sqrt((mass(8,2) - mass(9,2))^2+(mass(8,3) - mass(9,3))^2+(mass(8,4) - mass(9,4))^2),
9, 8;
          k, sqrt((mass(7,2) - mass(10,2))^2+(mass(7,3) - mass(10,3))^2+(mass(7,4) -
mass(10,4))^2), 10, 7;
          k, sqrt((mass(8,2) - mass(10,2))^2+(mass(8,3) - mass(10,3))^2+(mass(8,4) -
mass(10,4))^2), 10, 8;
          k, sqrt((mass(9,2) - mass(10,2))^2+(mass(9,3) - mass(10,3))^2+(mass(9,4) -
mass(10,4))^2), 10, 9;
          k, sqrt((mass(8,2) - mass(11,2))^2+(mass(8,3) - mass(11,3))^2+(mass(8,4) -
mass(11,4))^2), 11, 8;
          k, sqrt((mass(9,2) - mass(11,2))^2+(mass(9,3) - mass(11,3))^2+(mass(9,4) -
mass(11,4))^2), 11, 9;
          k, sqrt((mass(10,2) - mass(11,2))^2+(mass(10,3) - mass(11,3))^2+(mass(10,4) -
mass(11,4))^2), 11, 10;
          k, sqrt((mass(9,2) - mass(12,2))^2+(mass(9,3) - mass(12,3))^2+(mass(9,4) -
mass(12,4))^2), 12, 9;
          k, sqrt((mass(10,2) - mass(12,2))^2+(mass(10,3) - mass(12,3))^2+(mass(10,4) -
mass(12,4))^2), 12, 10;
          k, sqrt((mass(11,2) - mass(12,2))^2+(mass(11,3) - mass(12,3))^2+(mass(11,4) -
mass(12,4))^2), 12, 11;
          k, sqrt((mass(10,2) - mass(13,2))^2+(mass(10,3) - mass(13,3))^2+(mass(10,4) -
mass(13,4))^2), 13, 10;
          k, sqrt((mass(11,2) - mass(13,2))^2+(mass(11,3) - mass(13,3))^2+(mass(11,4) -
mass(13,4))^2), 13, 11;
          k, sqrt((mass(12,2) - mass(13,2))^2+(mass(12,3) - mass(13,3))^2+(mass(12,4) -
mass(13,4))^2), 13, 12;
          k, sqrt((mass(11,2) - mass(14,2))^2+(mass(11,3) - mass(14,3))^2+(mass(11,4) -
mass(14,4))^2), 14, 11;
          k, sqrt((mass(12,2) - mass(14,2))^2+(mass(12,3) - mass(14,3))^2+(mass(12,4) -
mass(14,4))^2), 14, 12;
          k, sqrt((mass(13,2) - mass(14,2))^2+(mass(13,3) - mass(14,3))^2+(mass(13,4) -
mass(14,4))^2), 14, 13;
          k, sqrt((mass(12,2) - mass(15,2))^2+(mass(12,3) - mass(15,3))^2+(mass(12,4) -
mass(15,4))^2), 15, 12;
          k, sqrt((mass(13,2) - mass(15,2))^2+(mass(13,3) - mass(15,3))^2+(mass(13,4) -
mass(15,4))^2), 15, 13;
          k, sqrt((mass(14,2) - mass(15,2))^2+(mass(14,3) - mass(15,3))^2+(mass(14,4) -
mass(15,4))^2), 15, 14;
          k, sqrt((mass(13,2) - mass(16,2))^2+(mass(13,3) - mass(16,3))^2+(mass(13,4) -
mass(16,4))^2), 16, 13;
          k, sqrt((mass(14,2) - mass(16,2))^2+(mass(14,3) - mass(16,3))^2+(mass(14,4) -
mass(16,4))^2), 16, 14;
          k, sqrt((mass(15,2) - mass(16,2))^2+(mass(15,3) - mass(16,3))^2+(mass(15,4) -
mass(16,4))^2), 16, 15;
          k, sqrt((mass(14,2) - mass(17,2))^2+(mass(14,3) - mass(17,3))^2+(mass(14,4) -
mass(17,4))^2), 17, 14;
          k, sqrt((mass(15,2) - mass(17,2))^2+(mass(15,3) - mass(17,3))^2+(mass(15,4) -
mass(17,4))^2), 17, 15;
          k, sqrt((mass(16,2) - mass(17,2))^2+(mass(16,3) - mass(17,3))^2+(mass(16,4) -
mass(17,4))^2), 17, 16;
          k, sqrt((mass(15,2) - mass(18,2))^2+(mass(15,3) - mass(18,3))^2+(mass(15,4) -
mass(18,4))^2), 18, 15;
          k, sqrt((mass(16,2) - mass(18,2))^2+(mass(16,3) - mass(18,3))^2+(mass(16,4) -
mass(18,4))^2), 18, 16;
          k, sqrt((mass(17,2) - mass(18,2))^2+(mass(17,3) - mass(18,3))^2+(mass(17,4) -
mass(18,4))^2), 18, 17;
          k, sqrt((mass(16,2) - mass(19,2))^2+(mass(16,3) - mass(19,3))^2+(mass(16,4) -
mass(19,4))^2), 19, 16;
          k, sqrt((mass(17,2) - mass(19,2))^2+(mass(17,3) - mass(19,3))^2+(mass(17,4) -
mass(19,4))^2), 19, 17;
          k, sqrt((mass(18,2) - mass(19,2))^2+(mass(18,3) - mass(19,3))^2+(mass(18,4) -
mass(19,4))^2), 19, 18;
          k, sqrt((mass(17,2) - mass(20,2))^2+(mass(17,3) - mass(20,3))^2+(mass(17,4) -
mass(20,4))^2), 20, 17;
```

```
          k, sqrt((mass(18,2) - mass(20,2))^2+(mass(18,3) - mass(20,3))^2+(mass(18,4) -
mass(20,4))^2), 20, 18;
          k, sqrt((mass(19,2) - mass(20,2))^2+(mass(19,3) - mass(20,3))^2+(mass(19,4) -
mass(20,4))^2), 20, 19;
          k, sqrt((mass(18,2) - mass(21,2))^2+(mass(18,3) - mass(21,3))^2+(mass(18,4) -
mass(21,4))^2), 21, 18;
          k, sqrt((mass(19,2) - mass(21,2))^2+(mass(19,3) - mass(21,3))^2+(mass(19,4) -
mass(21,4))^2), 21, 19;
          k, sqrt((mass(20,2) - mass(21,2))^2+(mass(20,3) - mass(21,3))^2+(mass(20,4) -
mass(21,4))^2), 21, 20;
          k, sqrt((mass(19,2) - mass(22,2))^2+(mass(19,3) - mass(22,3))^2+(mass(19,4) -
mass(22,4))^2), 22, 19;
          k, sqrt((mass(20,2) - mass(22,2))^2+(mass(20,3) - mass(22,3))^2+(mass(20,4) -
mass(22,4))^2), 22, 20;
          k, sqrt((mass(21,2) - mass(22,2))^2+(mass(21,3) - mass(22,3))^2+(mass(21,4) -
mass(22,4))^2), 22, 21;
          k, sqrt((mass(20,2) - mass(23,2))^2+(mass(20,3) - mass(23,3))^2+(mass(20,4) -
mass(23,4))^2), 23, 20;
          k, sqrt((mass(21,2) - mass(23,2))^2+(mass(21,3) - mass(23,3))^2+(mass(21,4) -
mass(23,4))^2), 23, 21;
          k, sqrt((mass(22,2) - mass(23,2))^2+(mass(22,3) - mass(23,3))^2+(mass(22,4) -
mass(23,4))^2), 23, 22;
          k, sqrt((mass(21,2) - mass(24,2))^2+(mass(21,3) - mass(24,3))^2+(mass(21,4) -
mass(24,4))^2), 24, 21;
          k, sqrt((mass(22,2) - mass(24,2))^2+(mass(22,3) - mass(24,3))^2+(mass(22,4) -
mass(24,4))^2), 24, 22;
          k, sqrt((mass(23,2) - mass(24,2))^2+(mass(23,3) - mass(24,3))^2+(mass(23,4) -
mass(24,4))^2), 24, 23;
          k, sqrt((mass(22,2) - mass(25,2))^2+(mass(22,3) - mass(25,3))^2+(mass(22,4) -
mass(25,4))^2), 25, 22;
          k, sqrt((mass(23,2) - mass(25,2))^2+(mass(23,3) - mass(25,3))^2+(mass(23,4) -
mass(25,4))^2), 25, 23;
          k, sqrt((mass(24,2) - mass(25,2))^2+(mass(24,3) - mass(25,3))^2+(mass(24,4) -
mass(25,4))^2), 25, 24;];
end


function [C1,C2] = Cross_Spring(P1, P2, ranIndex)
%Author: Nattapong Ounanong
%   This function perform crossover of the spring rest length parameters
C2 = P1;
C1 = P2;

for i = 1:ranIndex
    C2(i+ranIndex) = P2(i+ranIndex);
    C1(i+ranIndex) = P1(i+ranIndex);
end
end


function [spring_property,sL] = Indirect_code(T,spring_store)
% This code will perform indirect coding to the mass-spring simulation in
% order to improve robot mobility
sL = size(spring_store,1);
Bone = [1:randi(sL/1.5)];
hMuscle = [Bone(end)+1:randi([Bone(end)+1,sL-8])];
sMuscle = [hMuscle(end)+1:randi([hMuscle(end)+1,sL-1])];
air = [sMuscle(end)+1:sL];

% randomy select number of spring that would be used to alter rest length
% num_spring = randperm(size(spring_store,1));
% sampling = randi([1,size(spring_store,1)]);
% mod_spring = num_spring([1:sampling]);
% num_spring = length(mod_spring);

num_spring = length(sL);
% generate index to material to randomly select spring property within the
% cube
spring_property = [];

for i = 1:length(Bone)
```

```
    spring_property(i,:) = T(1,:);
end
for i = length(Bone)+1:length(hMuscle)+length(Bone)
    spring_property(i,:) = T(2,:);
end
for i = length(hMuscle)+length(Bone)+1:length(sMuscle)+length(hMuscle)+length(Bone)
spring_property(i,:) = T(3,:);
end
for i =
length(sMuscle)+length(hMuscle)+length(Bone)+1:length(air)+length(sMuscle)+length(hMuscle)+length
(Bone)
spring_property(i,:) = T(4,:);
end
end
```