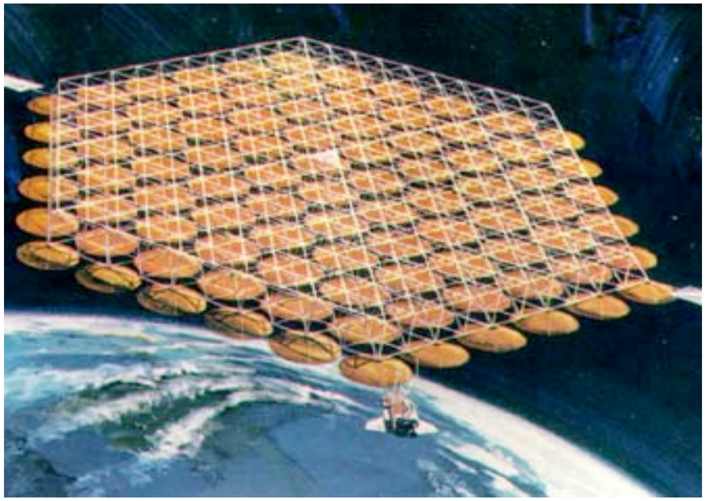
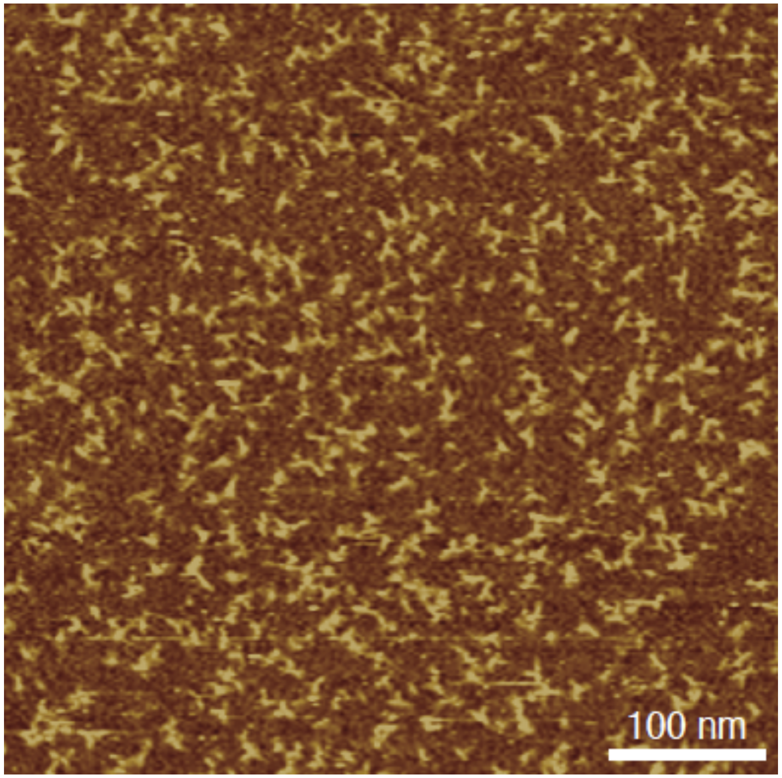
Self-Assembly of Three-Arm Junctions in DNA Strands

Sebastian Perusset
Francesco Bullo's Lab
Lab mentor: Anahita Mirabatabaei
Department of Mechanical Engineering

Big Picture

- Self Assembling systems:
  - the manufacture of a desired structure via autonomous behavior of the constituent parts
- Future Applications:
  - Simulate biological processes
  - Robotics
  - Aerospace



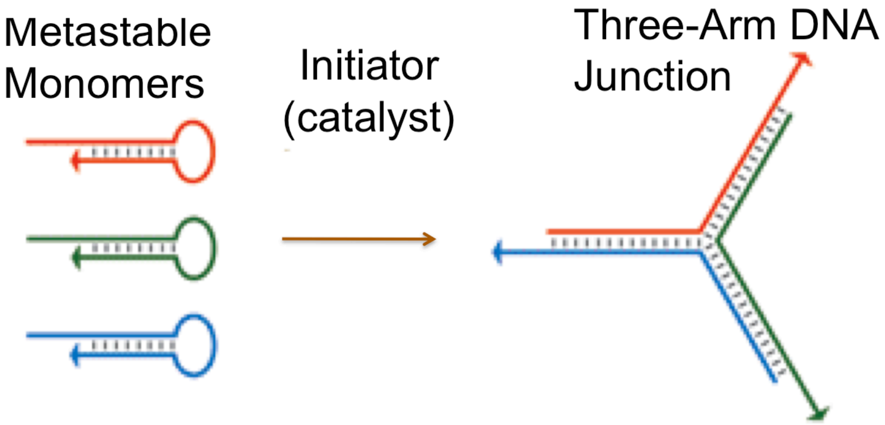


Peng, Yin "Programming Biomolecular Self-Assembly Pathways"

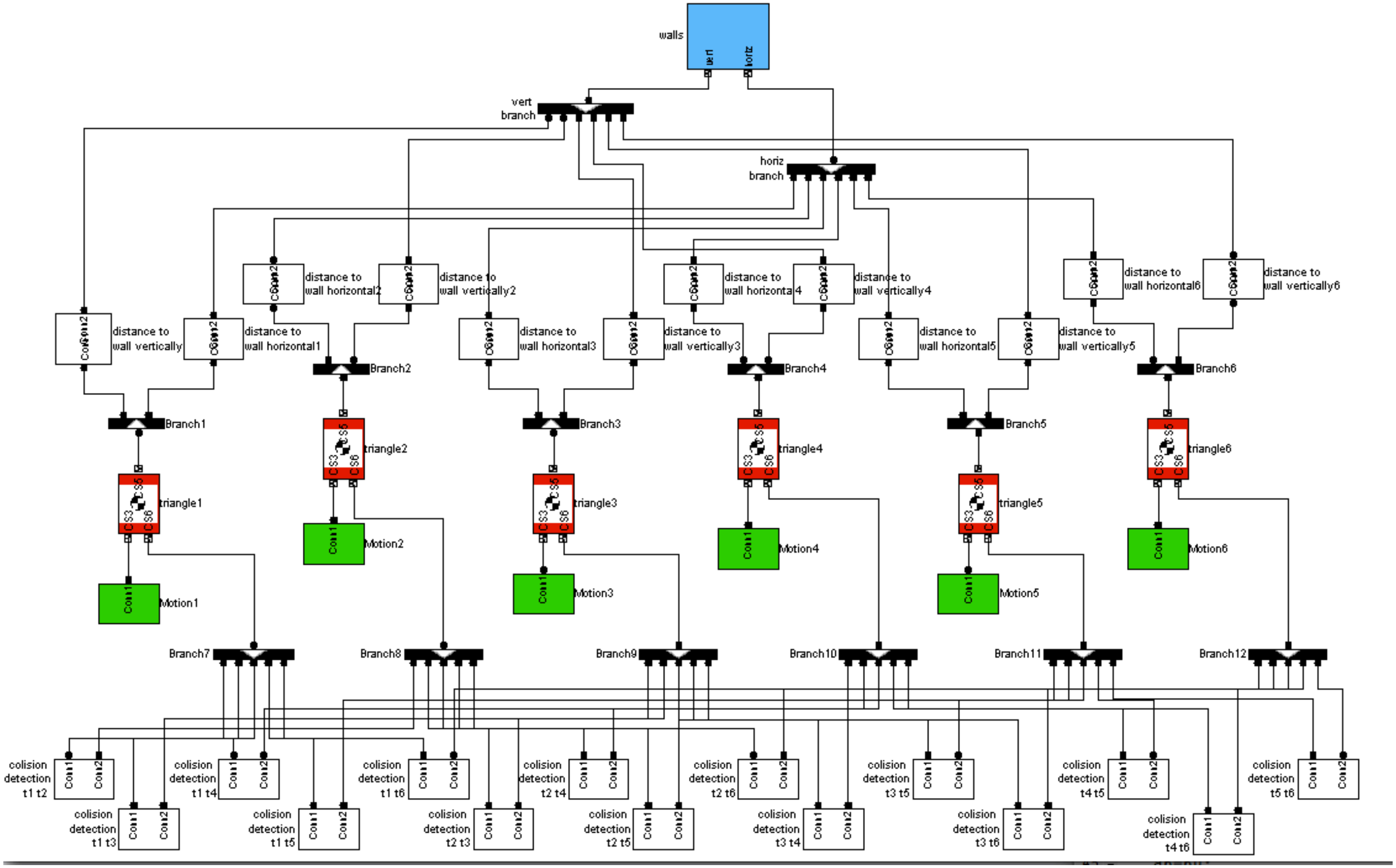
Project Goals

- Create Simulation
- Compute the time complexities and reaction graphs of the process
- Analyze the effect of catalysts
  - Using a collision probability factor
  - Simulating with real catalyst particles

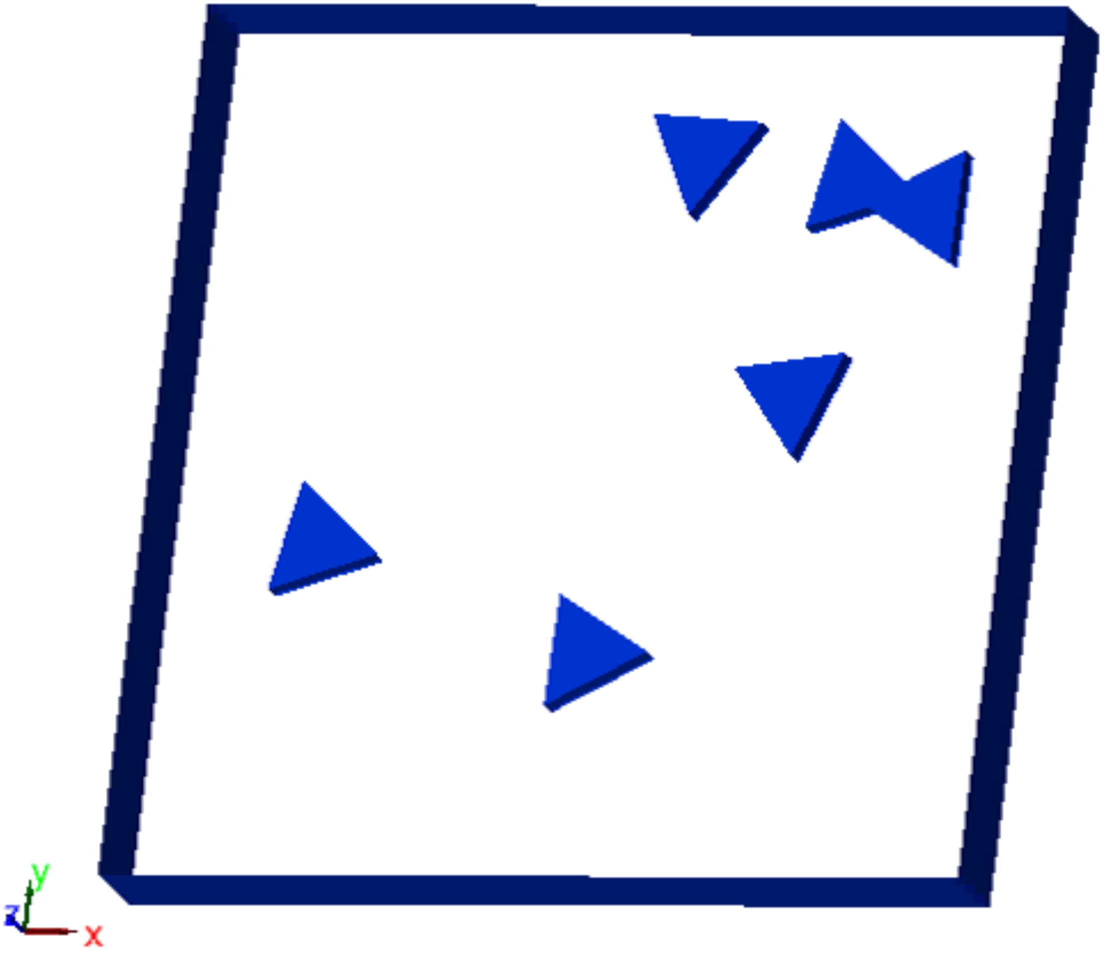
Three Arm Junctions in DNA Strands:



Simulink


walls
vert
horiz
vert branch
horiz branch
distance to wall vertically
distance to wall horizontal1
distance to wall horizontal2
distance to wall vertically2
distance to wall horizontal3
distance to wall vertically3
distance to wall horizontal4
distance to wall vertically4
distance to wall horizontal5
distance to wall vertically5
distance to wall horizontal6
distance to wall vertically6
Branch1
Branch2
Branch3
Branch4
Branch5
Branch6
triangle1
triangle2
triangle3
triangle4
triangle5
triangle6
Motion1
Motion2
Motion3
Motion4
Motion5
Motion6
Branch7
Branch8
Branch9
Branch10
Branch11
Branch12
colision detection t1 t2
colision detection t1 t3
colision detection t1 t4
colision detection t1 t5
colision detection t1 t6
colision detection t2 t3
colision detection t2 t4
colision detection t2 t5
colision detection t2 t6
colision detection t3 t4
colision detection t3 t5
colision detection t3 t6
colision detection t4 t5
colision detection t4 t6
colision detection t5 t6
Conn1
Conn2
CS3
CS5
CS6

Simulink Attempt



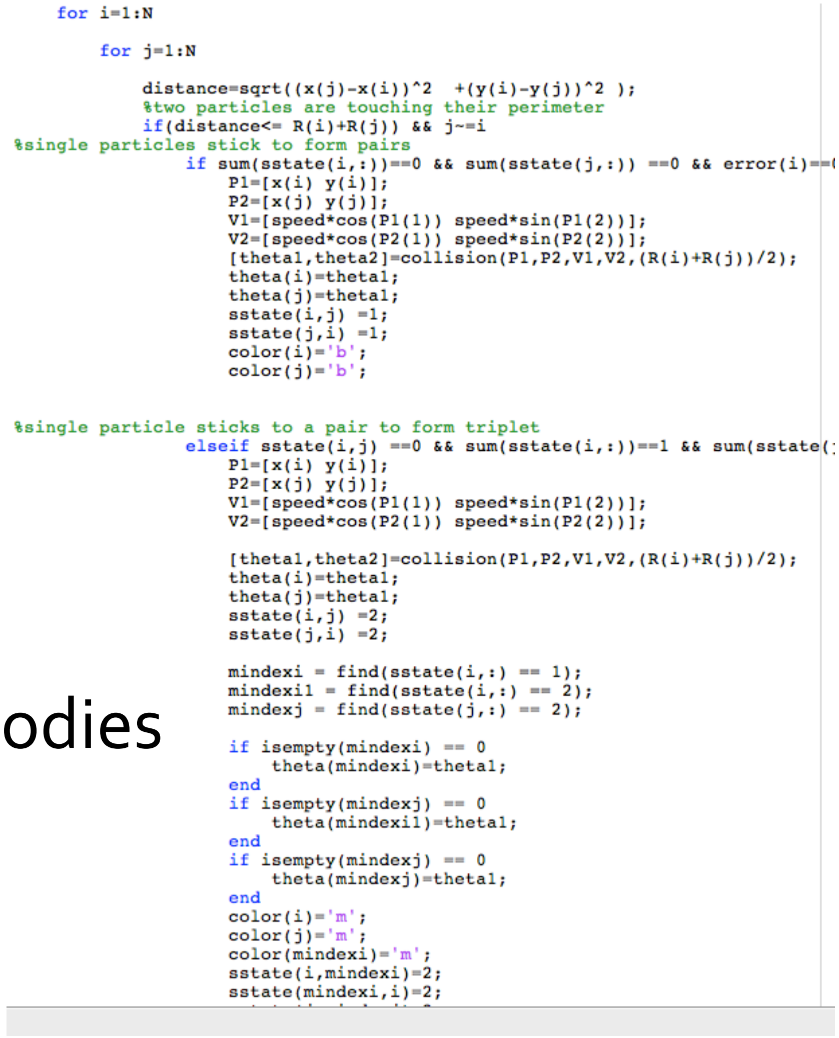
Matlab

Advantages:

- Faster
- More Flexible
- Less Redundancy
- Easy Access to Outputs

Disadvantages:

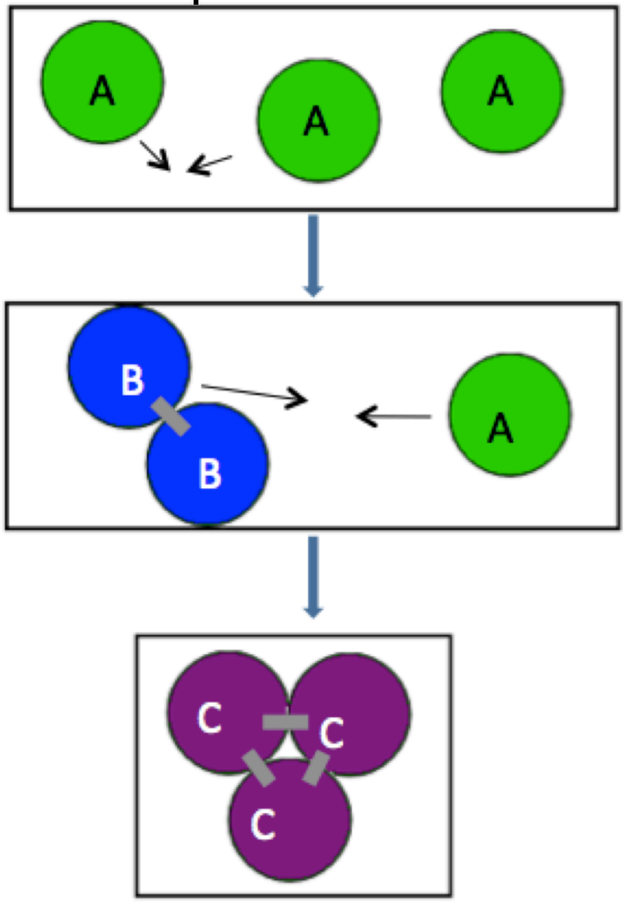
- Hard to create moving bodies

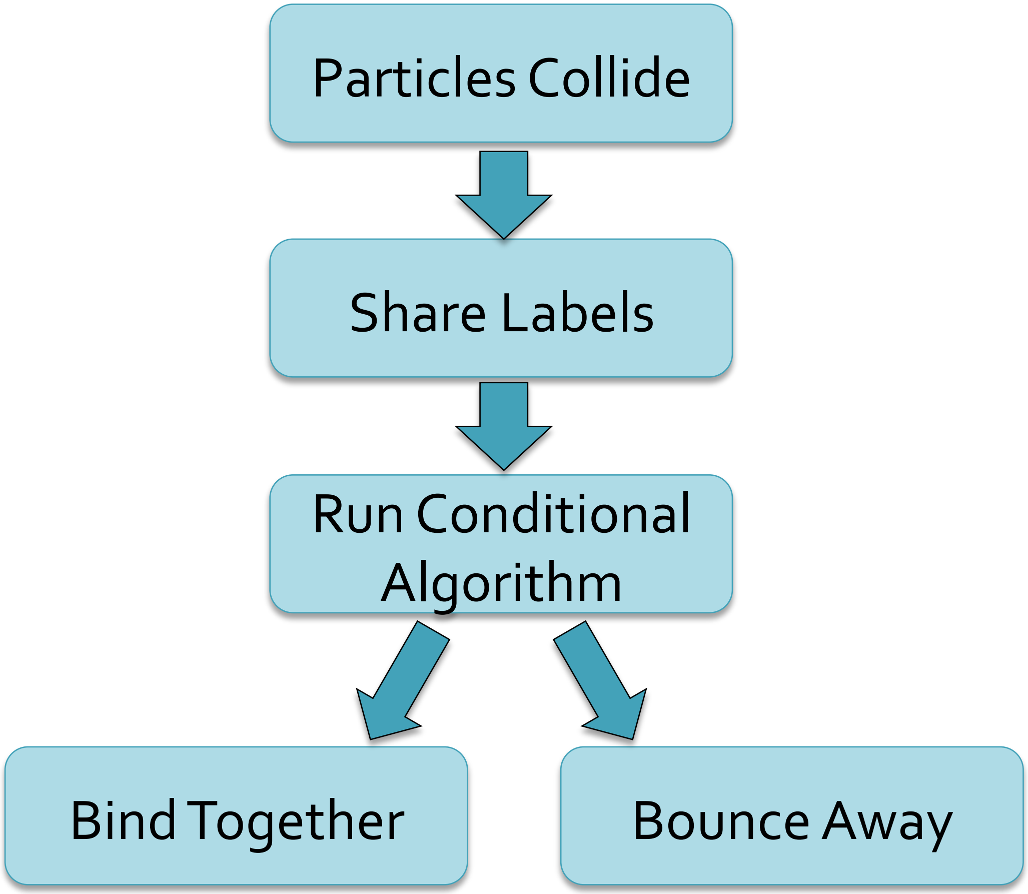


```
    for i=1:N

        for j=1:N

            distance=sqrt((x(j)-x(i))^2  +(y(i)-y(j))^2 );
            %two particles are touching their perimeter
            if(distance<= R(i)+R(j)) && j~=i
%single particles stick to form pairs
                if sum(sstate(i,:))==0 && sum(sstate(j,:)) ==0 && error(i)==
                    P1=[x(i) y(i)];
                    P2=[x(j) y(j)];
                    V1=[speed*cos(P1(1)) speed*sin(P1(2))];
                    V2=[speed*cos(P2(1)) speed*sin(P2(2))];
                    [theta1,theta2]=collision(P1,P2,V1,V2,(R(i)+R(j))/2);
                    theta(i)=theta1;
                    theta(j)=theta1;
                    sstate(i,j) =1;
                    sstate(j,i) =1;
                    color(i)='b';
                    color(j)='b';


%single particle sticks to a pair to form triplet
                elseif sstate(i,j) ==0 && sum(sstate(i,:))==1 && sum(sstate(
                    P1=[x(i) y(i)];
                    P2=[x(j) y(j)];
                    V1=[speed*cos(P1(1)) speed*sin(P1(2))];
                    V2=[speed*cos(P2(1)) speed*sin(P2(2))];

                    [theta1,theta2]=collision(P1,P2,V1,V2,(R(i)+R(j))/2);
                    theta(i)=theta1;
                    theta(j)=theta1;
                    sstate(i,j) =2;
                    sstate(j,i) =2;

                    mindexi = find(sstate(i,:) == 1);
                    mindexi1 = find(sstate(i,:) == 2);
                    mindexj = find(sstate(j,:) == 2);

                    if isempty(mindexi) == 0
                        theta(mindexi)=theta1;
                    end
                    if isempty(mindexj) == 0
                        theta(mindexi1)=theta1;
                    end
                    if isempty(mindexj) == 0
                        theta(mindexj)=theta1;
                    end
                    color(i)='m';
                    color(j)='m';
                    color(mindexi)='m';
                    sstate(i,mindexi)=2;
                    sstate(mindexi,i)=2;
```

How it works

Graph Grammar





Particles Collide

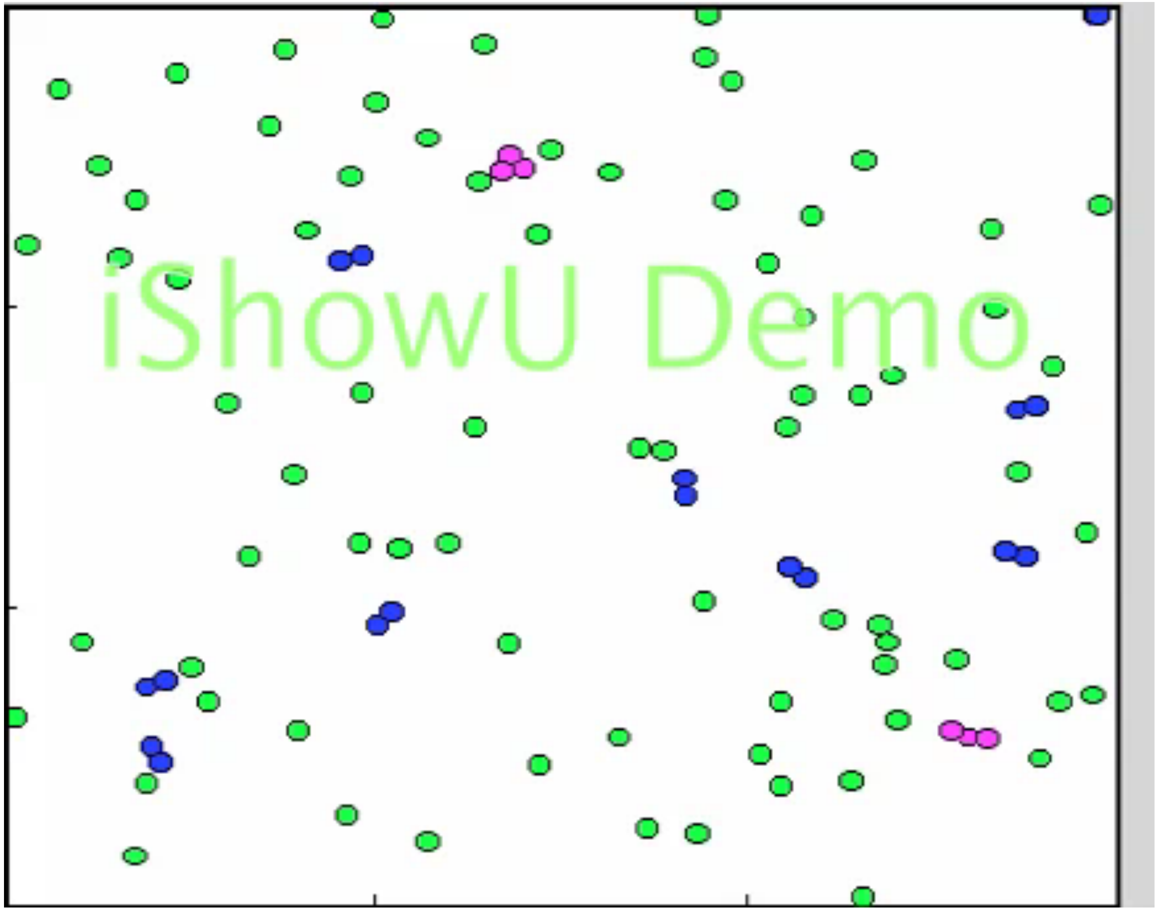
Share Labels

Run Conditional Algorithm

Bind Together

Bounce Away

Simulation With Probability Factor



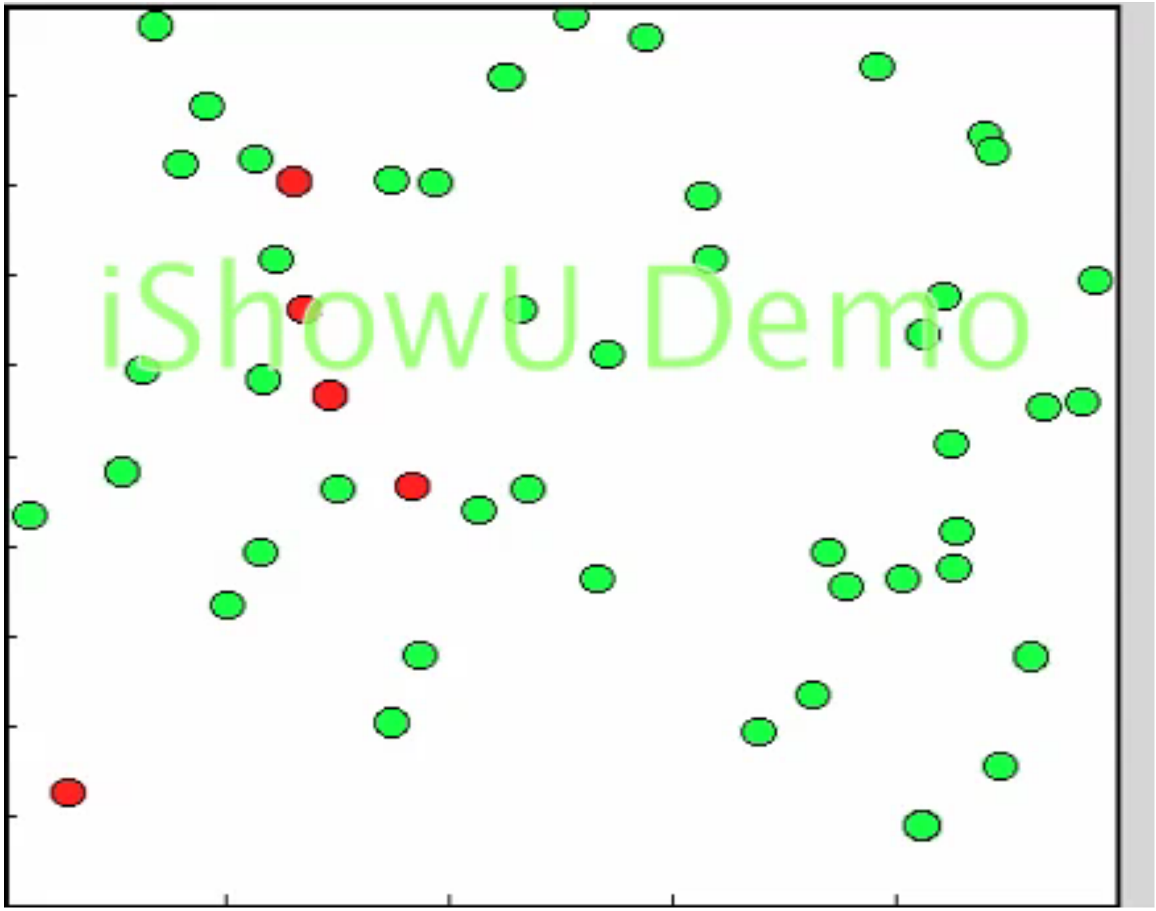
Legend:

● --Monomer

● --Unstable Dimer

● --Three Arm Junction

Simulation With Initiator Particles



Legend:

🟢 --Monomer

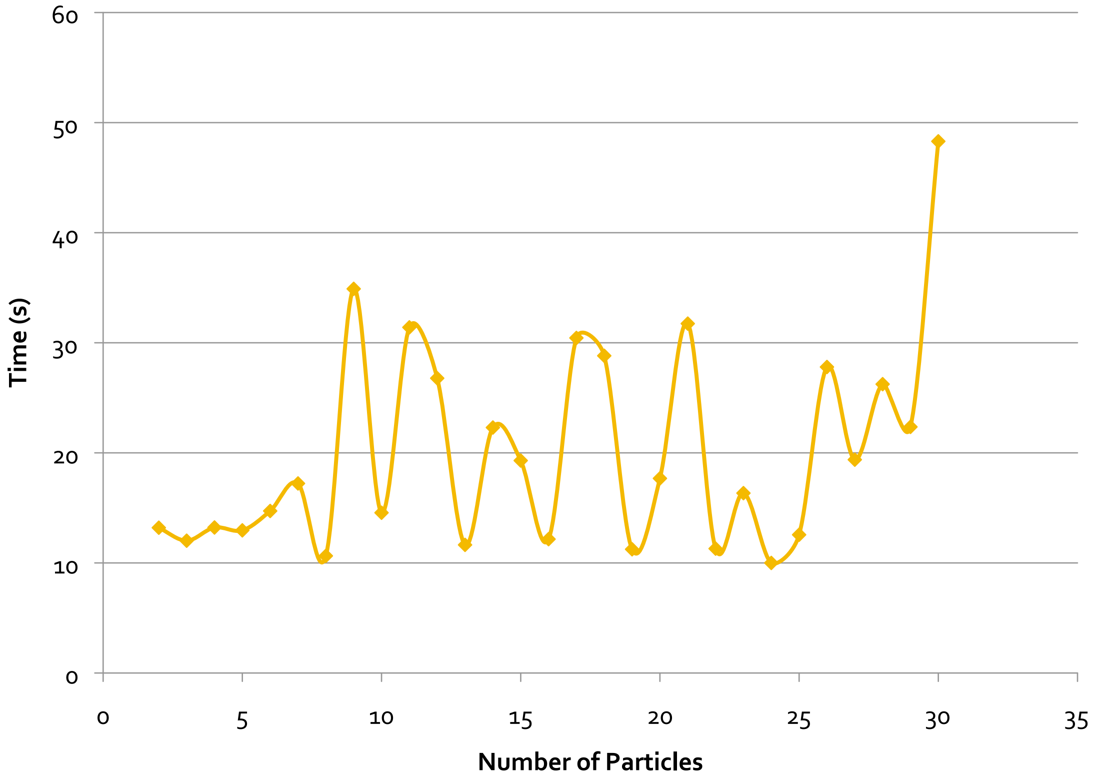
🔴 --Catalyst

🟡⚫ --Catalyst and Monomer

🔵🔵 --Unstable Dimer

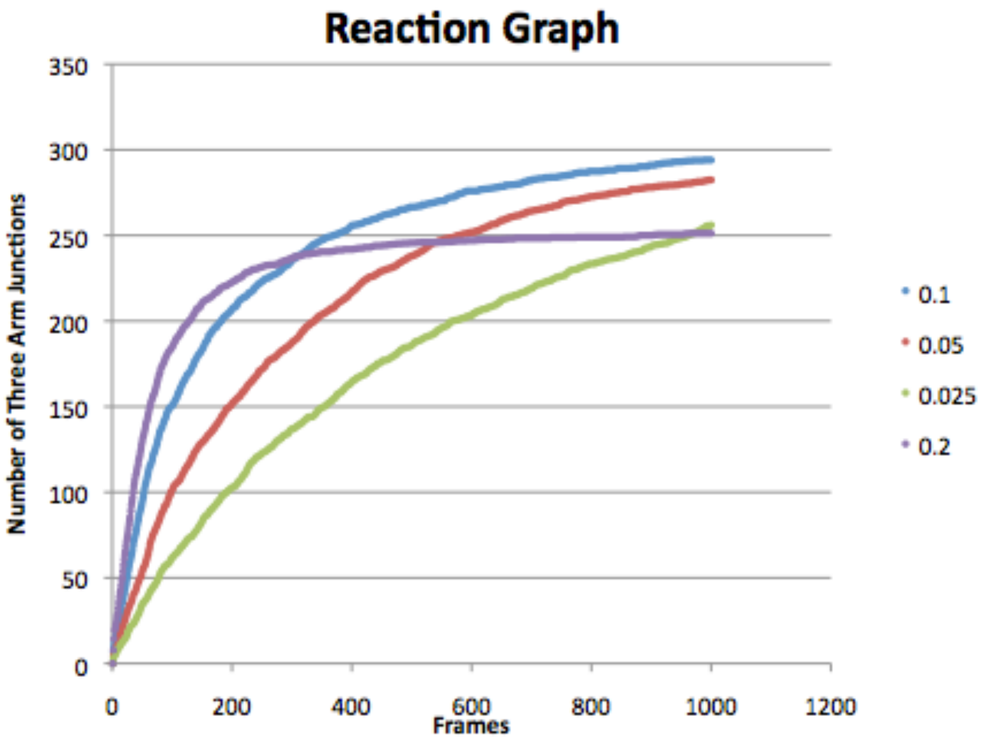
🟣🟣🟣 --Three Arm Junction

Time Complexity Graph

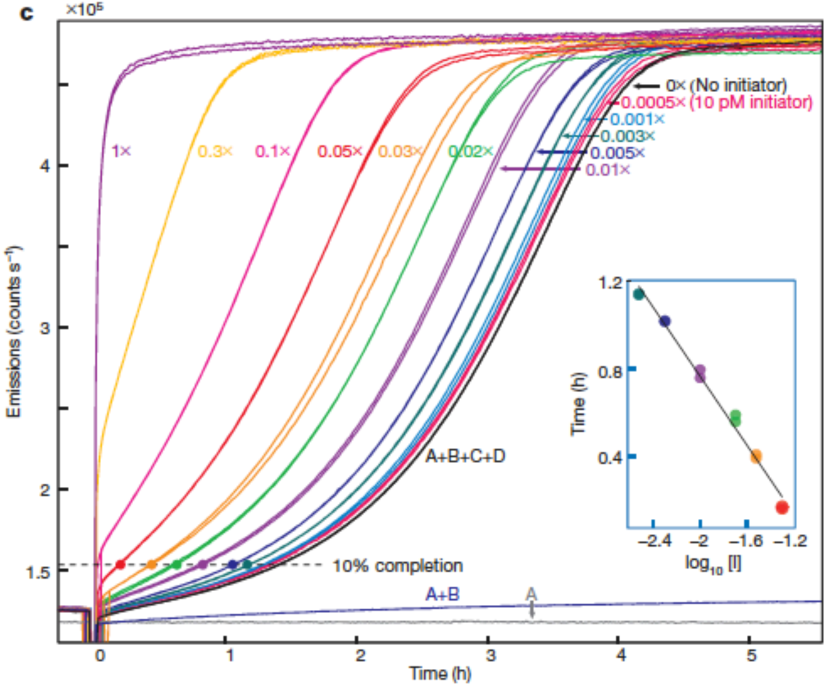


- Not much correlation
- Pattern of threes

Conclusions



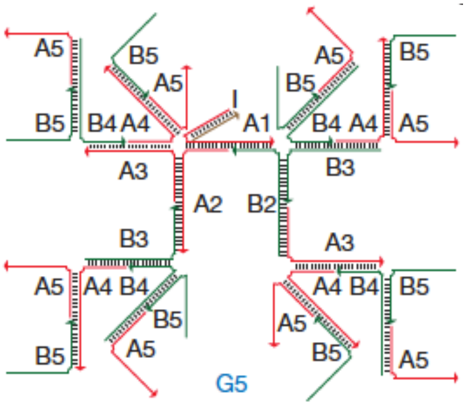
Experimental Reaction Graph



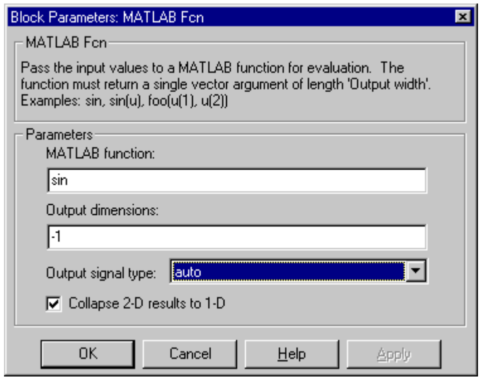
- More catalyst = Faster completion time
- Too much catalyst will result in less final product

Future Directions

- Simulate more complex self-assemblies



- Make simulation more user friendly



Block Parameters: MATLAB Fcn

MATLAB Fcn

Pass the input values to a MATLAB function for evaluation. The function must return a single vector argument of length 'Output width'. Examples: sin, sin(u), foo(u(1), u(2))

Parameters

MATLAB function: sin

Output dimensions: -1

Output signal type: auto

☑ Collapse 2-D results to 1-D

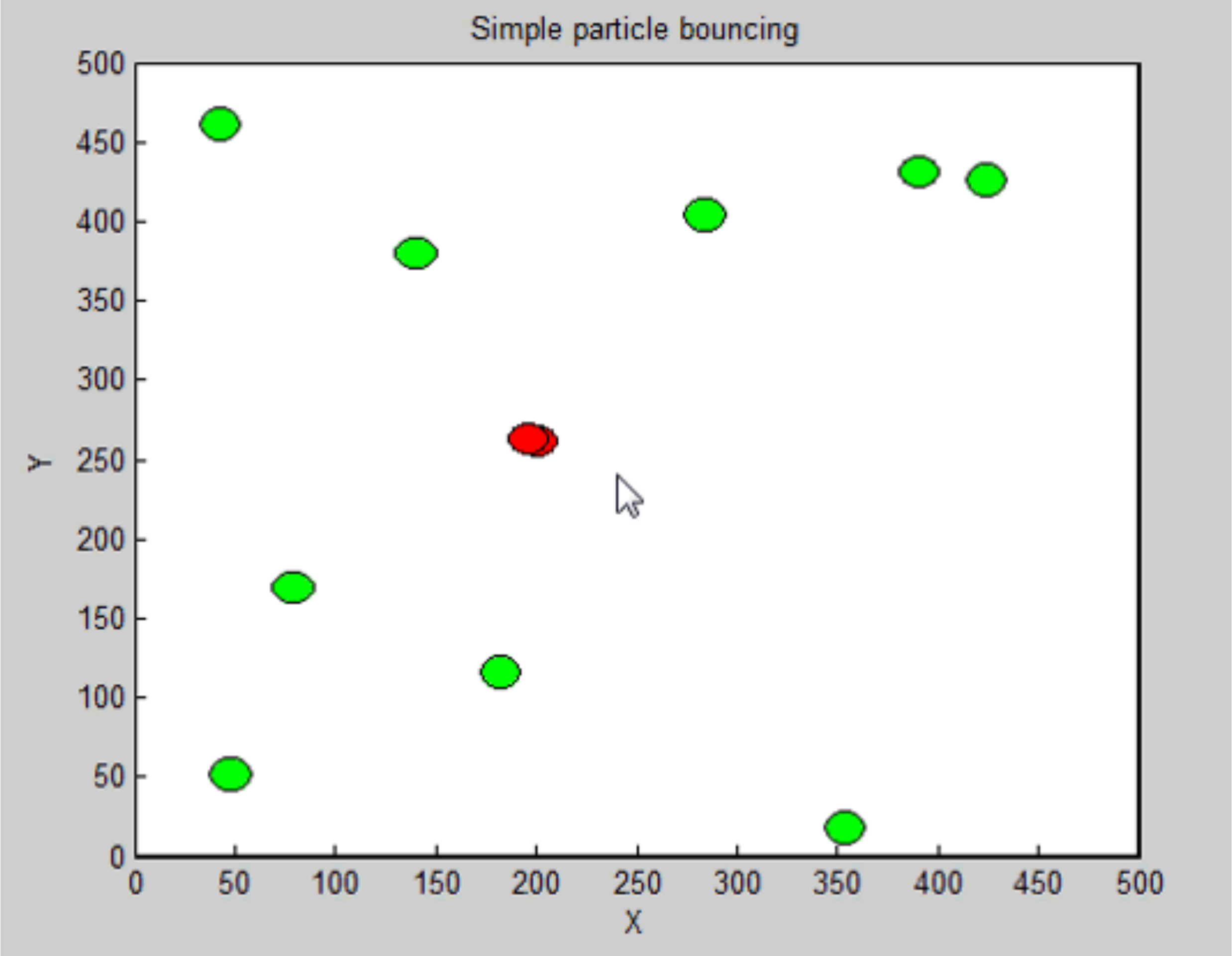
- Create software for self-assembling robots

Questions?

Special thank you to:

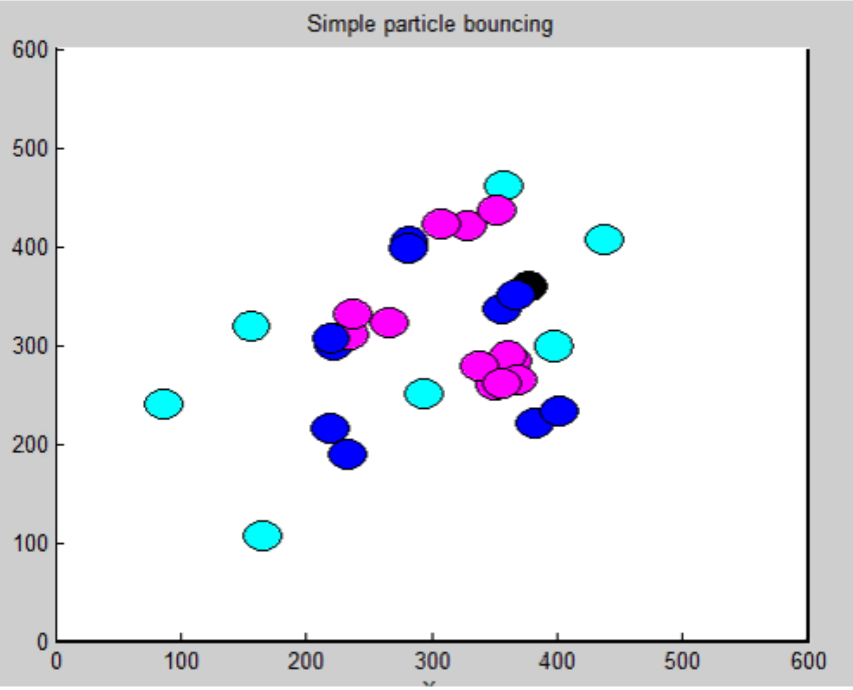
- Arica Lubin
- Francesco Bullo
- Anahita Mirabatabaei

Matlab Progress

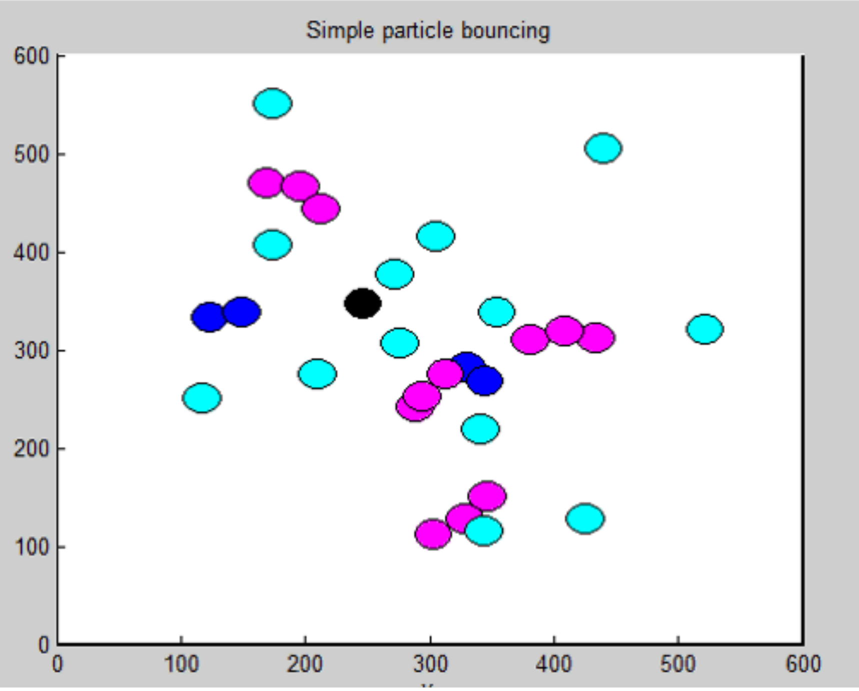


Malfunctioning Agents

Does not stick to any



Destroys all bonds


Simple particle bouncing