Giotto:
A painter robot

Robotics and Humans interfaces project

Davide Rasla - Embedded Computing Systems

01.

The idea

What Giotto can do

02.

Possible applications

Examples of some paintings

03.

Image processing

How to obtain a trajectory from a image

04.

Physics

Simple analysis of real requirements

05.

Avoidance

Avoid an enemy

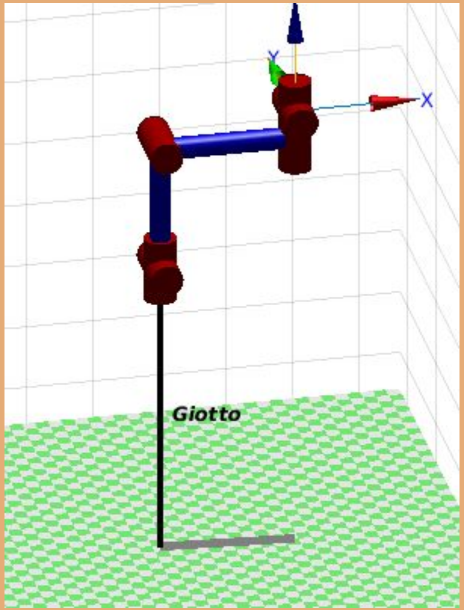
01

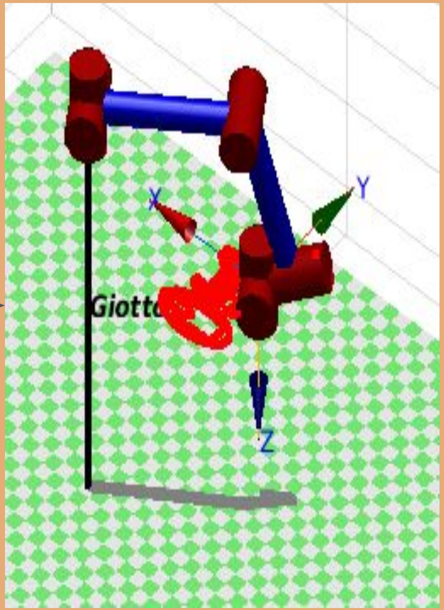
The Idea

What Giotto can do

Objective

The aim is to create a robot able to draw any kind of image on a plane

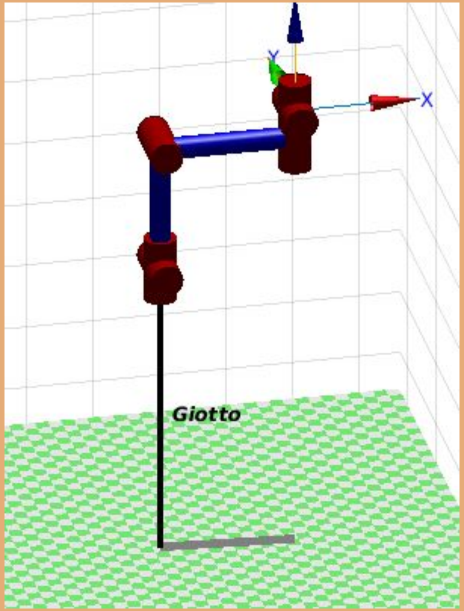


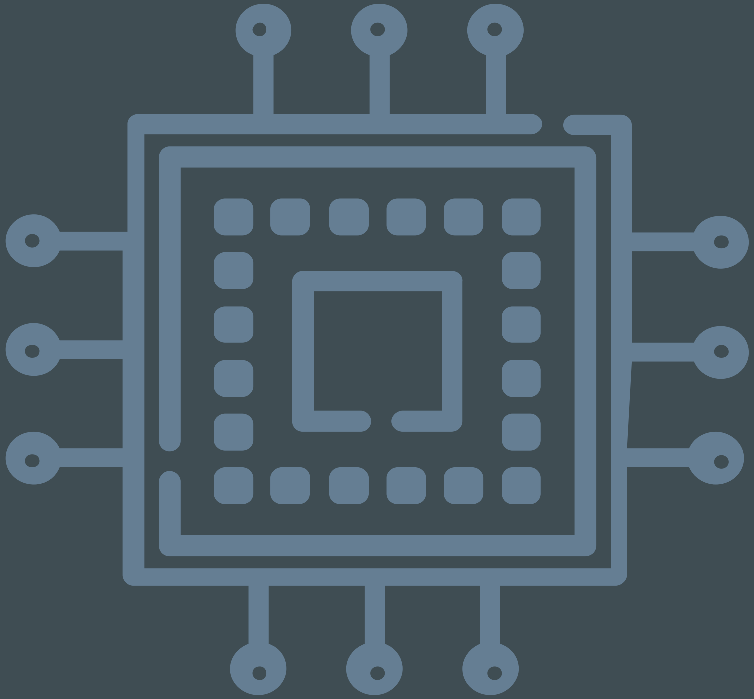


The idea

```
L6= Revolute('d', 0, 'a', 0, 'alpha', 0,  ...
    'I', [0.15e-3, 0.15e-3, 0.04e-3, 0, 0, 0], ...
    'r', [0, 0, 0.032], ...
    'm', 0.09, ...
    'Jm', 33e-6, ...
    'G', 76.686, ...
    'B', 36.7e-6, ...
    'Tc', [3.96e-3, -10.5e-3], ...
    'qlim', [-266 266]*deg );
qz = [0 0 0 0 0 0];
qr = [0 pi/2 -pi/2 0 0 0]; % ready pose, arm up
qs = [0 0 -pi/2 0 0 0];

L = [L1 L2 L3 L4 L5 L6];
rob = SerialLink(L , 'name', 'Giotto');
```





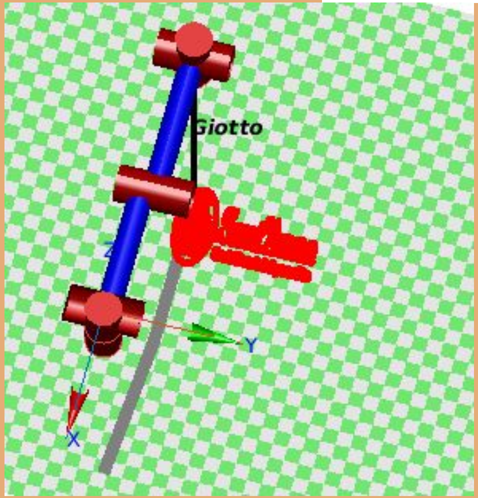
02

Possible applications

Examples of some paintings

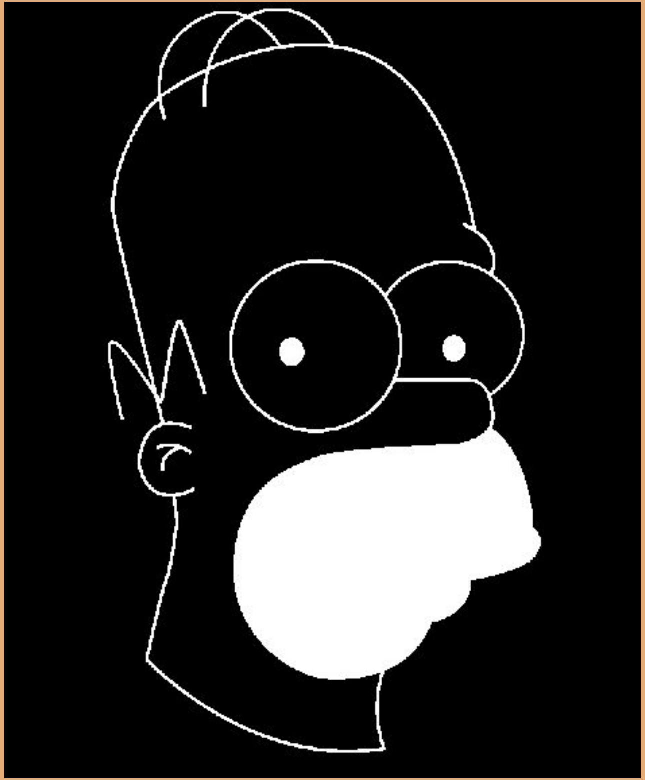
Applications

Sant'Anna
Scuola Universitaria Superiore Pisa

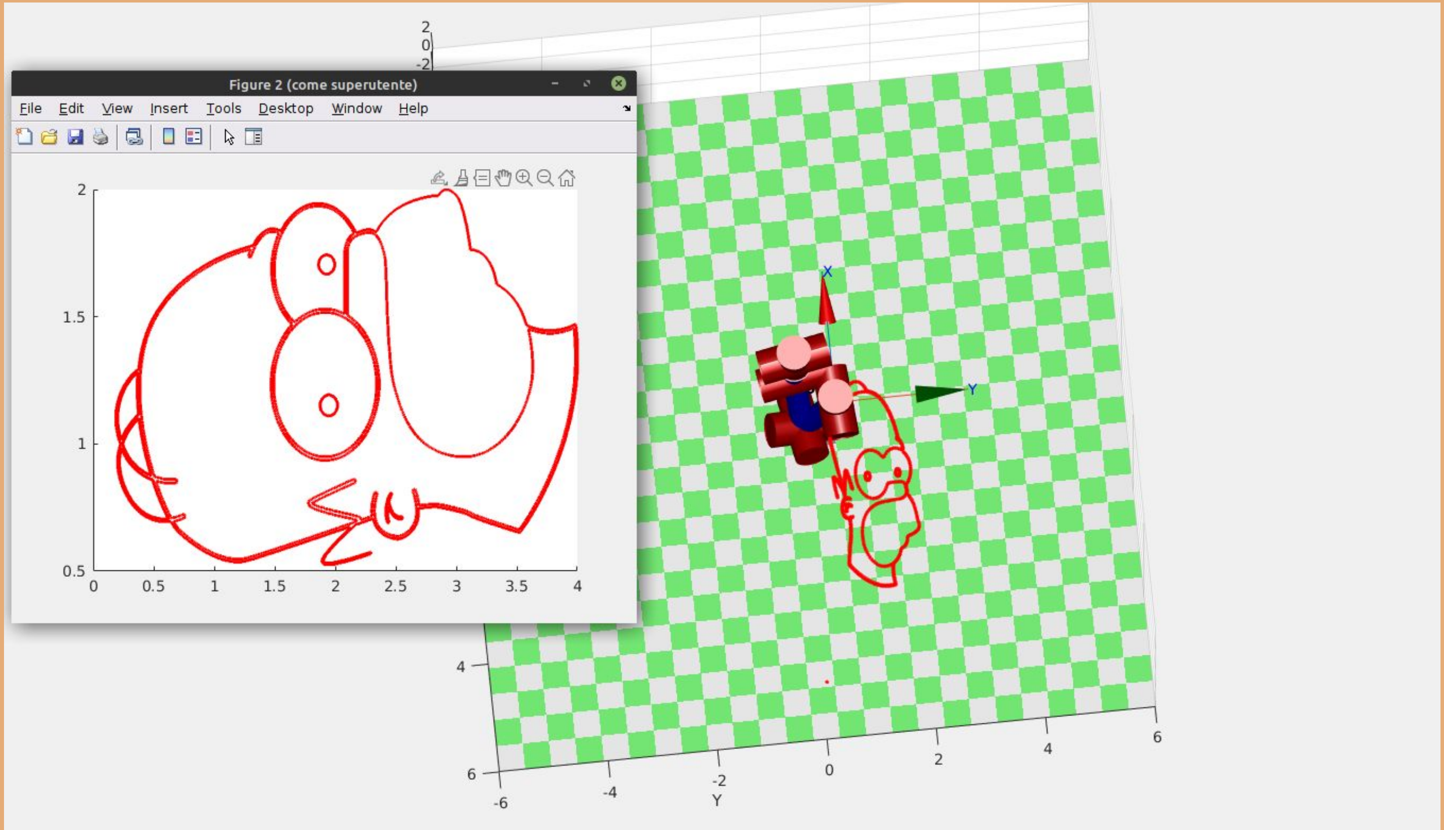


Applications

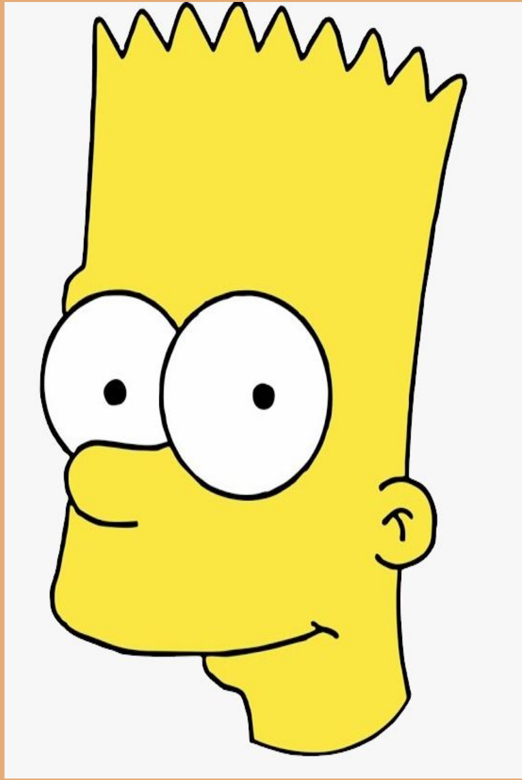
Applications

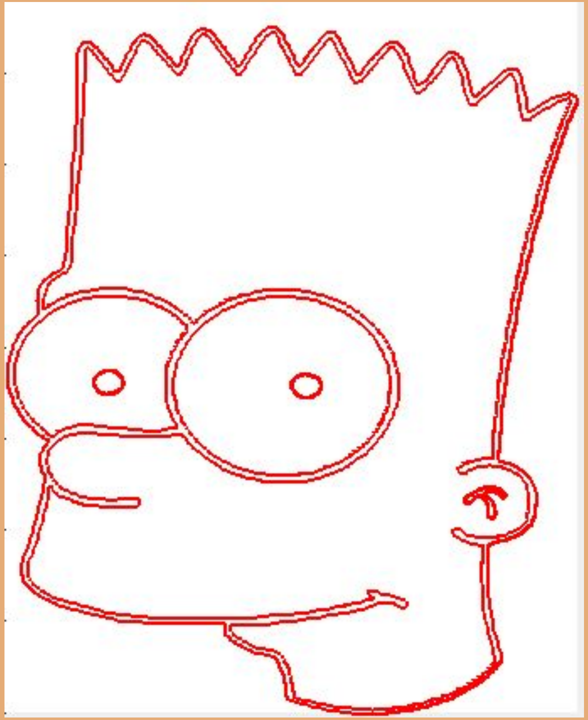


Applications

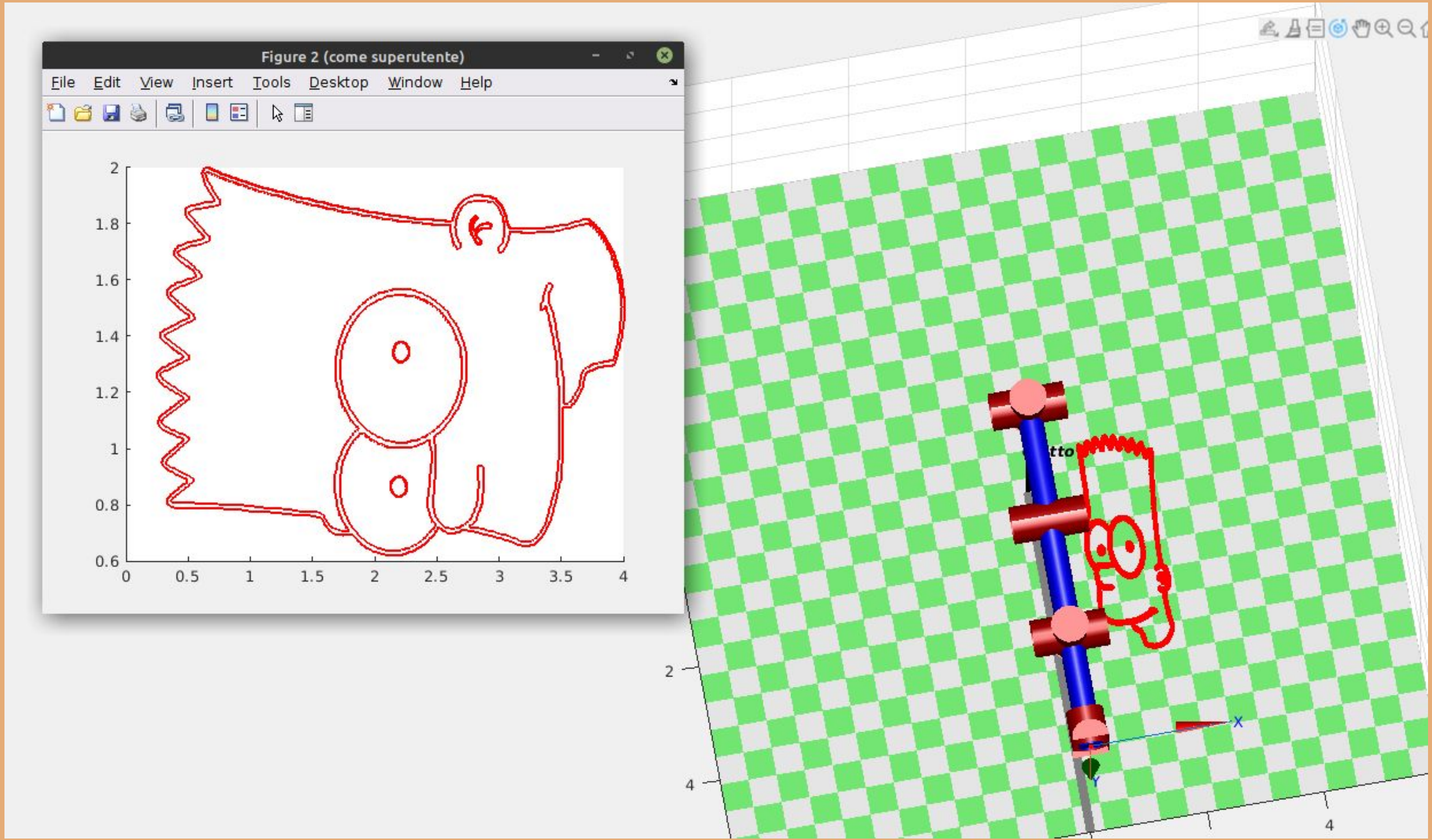


Applications

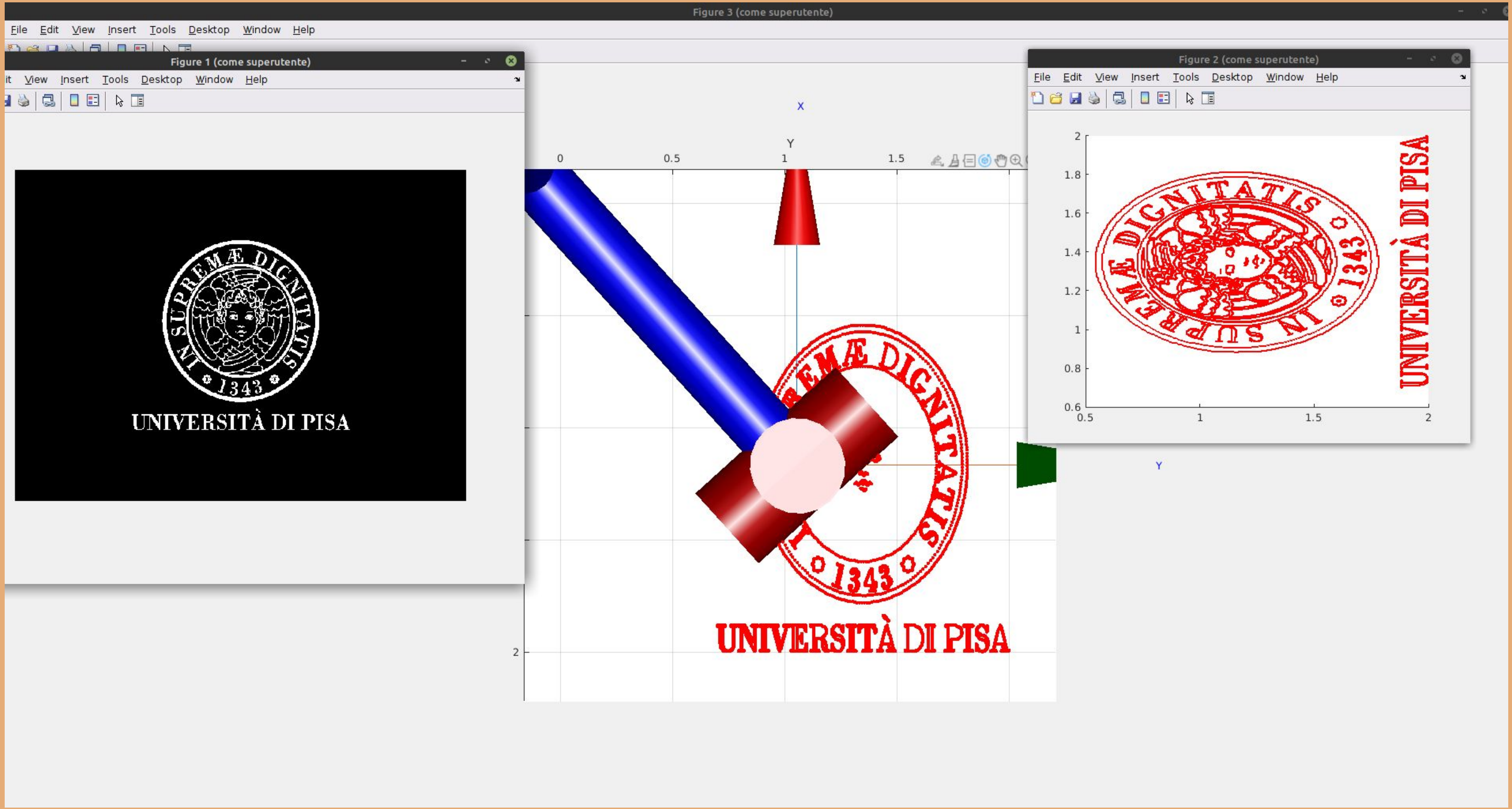




Applications



Applications

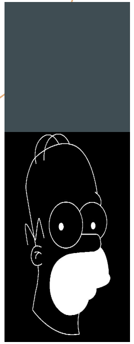


03

Image Processing

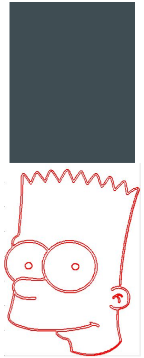
How to get a trajectory from any image

Processing Phases



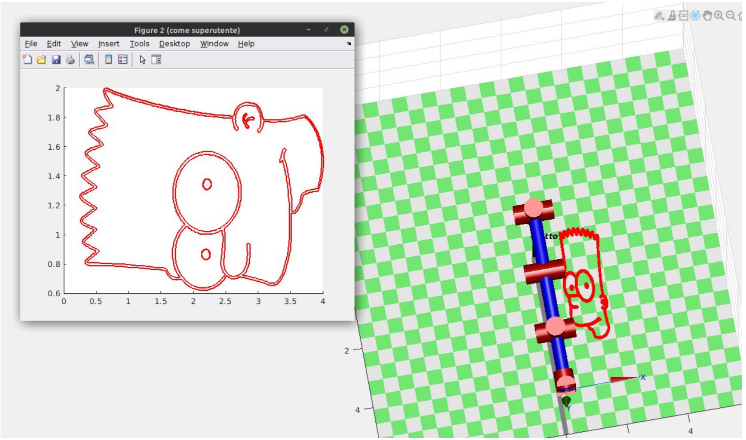
Phase 1

Obtain a binary BW image



Phase 2

Get the trajectory as a cell array



Phase 3

Rescale in proportion to the arm

Phase 4

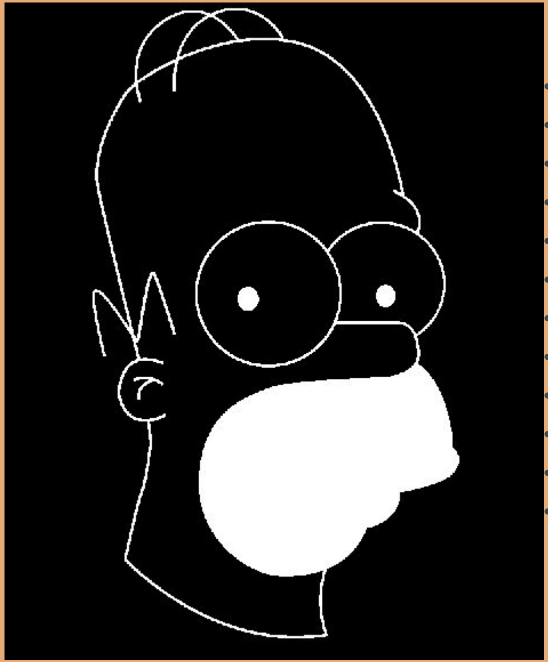
Filtering and apply inverse kinematics

Phase 1: Binary image

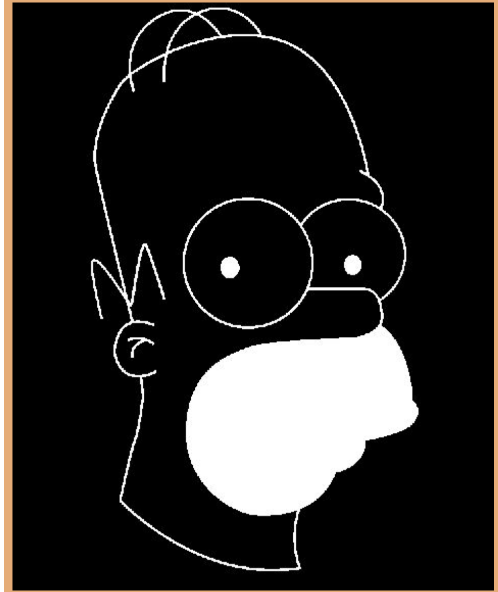
```
rgbImage = imread('Images/Homer.png');

greenChannel = rgbImage(:, :, 2);

binaryImage = greenChannel < 200;
```



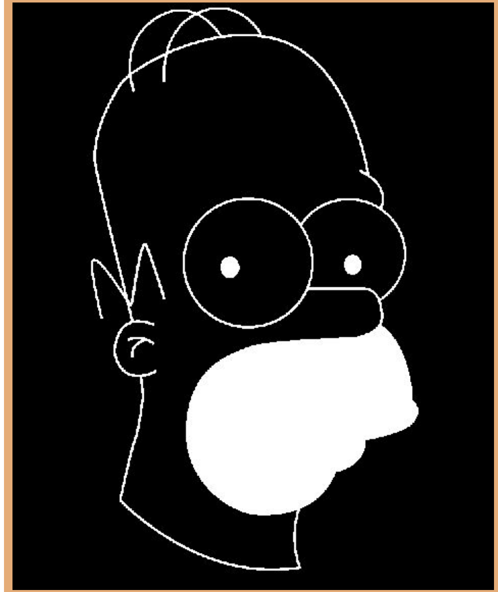
Phase 2: Getting Trajectory



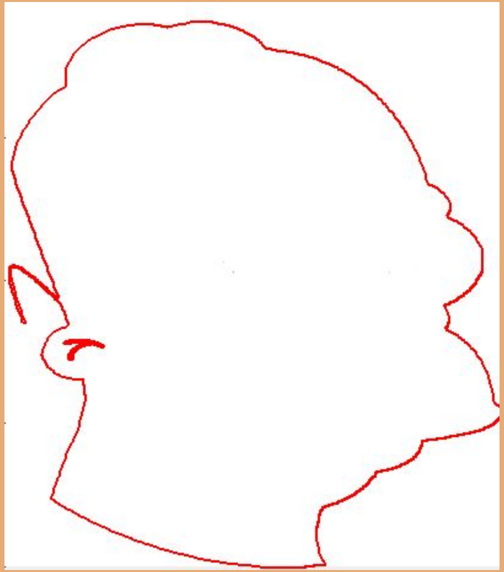
Holes features requires Several different trajectory. Code too long to show here

```
[B,L,N, A] = bwboundaries(binaryImage,4,'holes');%use 'noholes'
                                                 %to simplify
```

Phase 2: Getting Trajectory



No-Holes simplify the draw, but usually it is not necessary at all



```
[B,L,N, A] = bwboundaries(binaryImage,4,'noholes');%use 'noholes'
                                                    %to simplify
```

Phase 3: Rescale

Orginal coordinates too large for the arm. A rescale in [0: ARM_LENGTH] is necessary

```
%Manual rescaling: Getting the largest values of them all
    maxX = 0;
    maxY = 0;
    for i=1:length(max_Arrays)
        if max_Arrays{i}(1) > maxX
            maxX = max_Arrays{i}(1);
        end
         if max_Arrays{i}(2) > maxY
            maxY = max_Arrays{i}(2);
        end
    end
%Rescaling each traj by the max values of all, for each axis, multipling
%by two in order to obtain [0;2]

     for k =1:length(trajectories)
         for i = 1:length(trajectories{k})
          trajectories{k}(i,1) = trajectories{k}(i,1) / maxX * 2;%dividing maxX
          trajectories{k}(i,2) = trajectories{k}(i,2) / maxY * 2;%dividing maxY
         end
         plot(trajectories{k}(:,1), trajectories{k}(:,2), 'r', 'LineWidth', 2);
     end
```

Phase 4: Filtering

Orginal points are too close each others and are uncessary in most of cases. It can be useful to filter the coordinates

```
function trajectories = Filtering(trajectories)
    V = [];
    for i = 1:length(trajectories)
        if length(trajectories{i}) > 3500
            for j = 1:30:length(trajectories{i})
              V = [V; trajectories{i}(j,:)];
            end
            trajectories{i} = V;
        elseif length(trajectories{i}) > 1500
            for j = 1:5:length(trajectories{i})
              V = [V; trajectories{i}(j,:)];
            end
            trajectories{i} = V;
        elseif length(trajectories{i}) > 400
            for j = 1:3:length(trajectories{i})
              V = [V; trajectories{i}(j,:)];
            end
            trajectories{i} = V;
        end
        V = [];
    end
end
```

04

Physics

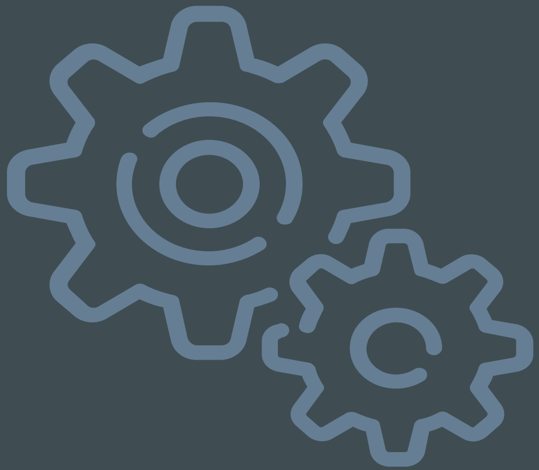
Inertia and other parameters

Parameters

```
L2 = Revolute('d', 0, 'a', 2, 'alpha', 0, ...
    'I', [0.13, 0.524, 0.539, 0, 0, 0], ...
    'r', [-0.3638, 0.006, 0.2275], ...
    'm', 17.4, ...
    'Jm', 200e-6, ...
    'G', 107.815, ...
    'B', .817e-3, ...
    'Tc', [0.126 -0.071], ...
    'qlim', [-45 225]*deg );
```

In order to create a controller, some parameters are necessary. Puma560 was taken as an initial point.

- Link Mass: computed for each link
- Ratio, coefficient and others are supposed as given
- Inertia is computed assuming the link as a cylinder

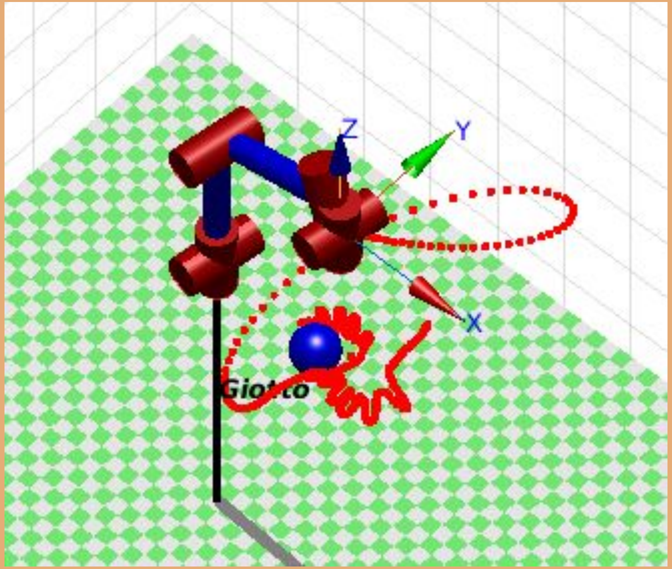


04

Avoidance

How to avoid an obs

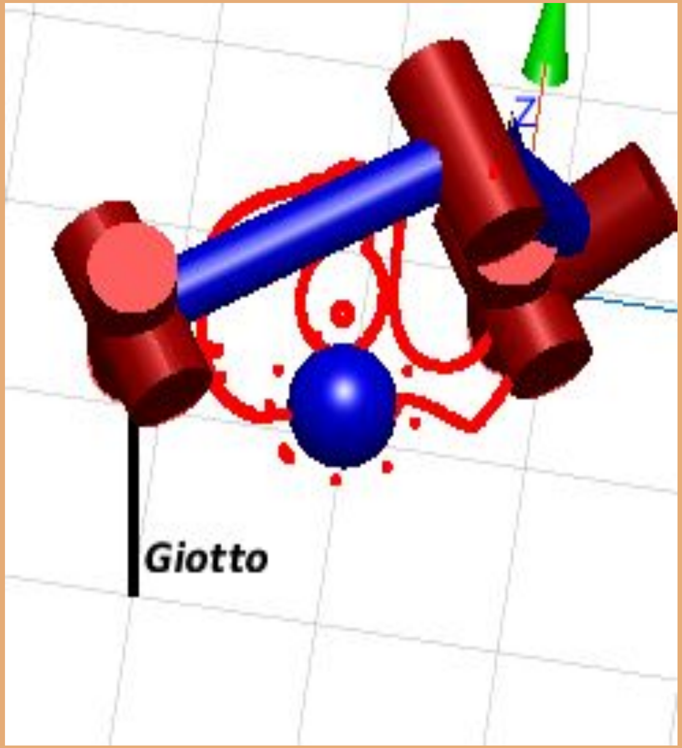
Features

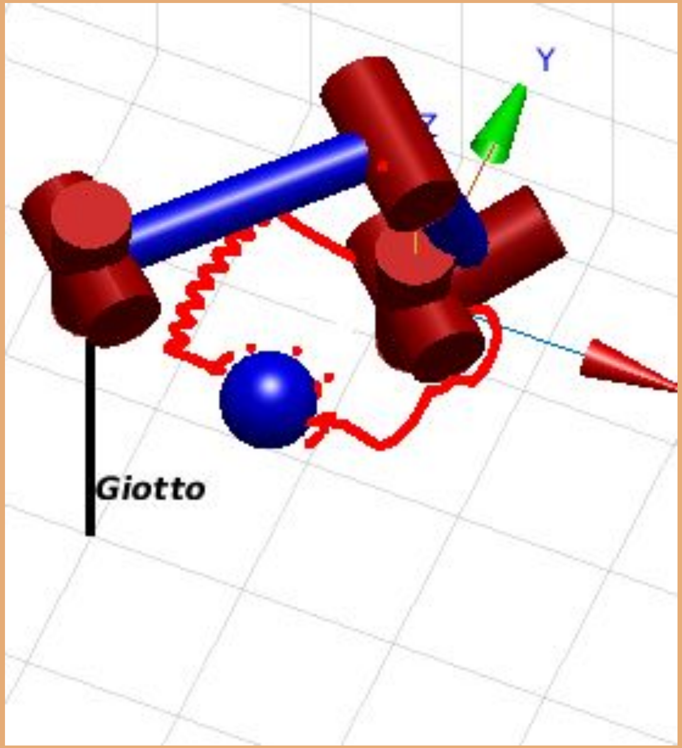


Using Optimization techniques a simple avoidance obstacle has been implemented moving the trajectory away from an obs minimizing the distance from it

A real painter would escape...a robot cannot!

Some examples





Control torque