Unified Robot Description Format (URDF)

Building Robots from Texts

Sungjoon Choi, Korea University

kakaobrain

URDF

ROBOT INTELLIGENCE LAB

- **U**nified **R**obot **D**escription **F**ormat (URDF)
  - An xml format for representing a robot model
  - Note that not all xml files are formatted with URDF
  - Other more complicated formats exists (e.g., mjcf for MuJoCo)

- In this lecture, we will be focussing on URDF as it is the **simplest** format to describe a robot model.
- We will parse the **core parts** of URDF to construct the kinematic chain.
- Also, we will learn about the capsule representation of meshes for collision checking.

URDF

ROBOT INTELLIGENCE LAB

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
```

```
<!--joint between base and link_0-->
<joint name="base_iiwa7_joint" type="fixed">
    <parent link="base"/>
    <child link="iiwa7_link_0"/>
    <origin rpy="0.0 0.0 0.0" xyz="0.0 0.0 0.0"/>
</joint>
```

Joint

```
<link name="iiwa7_link_0">
    <inertial>
        <origin xyz="-0.1 0 0.07" rpy="0 0 0"/>
        <mass value="5"/>
        <inertia ixx="0.05"  ixy="0"  ixz="0" iyy="0.06" iyz="0" izz="0.03" />
    </inertial>

    <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://iiwa_description/meshes/iiwa7/visual/link_0.stl"/>
        </geometry>
        <material name="Grey"/>
    </visual>

    <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://iiwa_description/meshes/iiwa7/collision/link_0.stl"/>
        </geometry>
        <material name="Grey"/>
    </collision>

    <self_collision_checking>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <capsule radius="0.15" length="0.25"/>
        </geometry>
    </self_collision_checking>
</link>
```

Link

Joint

ROBOT INTELLIGENCE LAB

```
<!-- joint between link_0 and link_1 -->
<joint name="iiwa7_joint_1" type="revolute">
    <parent link="iiwa7_link_0"/>
    <child link="iiwa7_link_1"/>
    <origin xyz="0 0 0.15" rpy="0 0 0"/>
    <axis xyz="0 0 1"/>
    <limit lower="-2.9671" upper="2.9671"
           effort="300" velocity="10" />
    <safety_controller soft_lower_limit="-2.9322"
                       soft_upper_limit="2.9322"
                       k_position="100"
                       k_velocity="2"/>
    <dynamics damping="0.5"/>
</joint>
```

- **Joint** field contains
  - Joint name and type of joint (fixed, revolute, prismatic, etc)
  - Names of the parent and child **links**. (one joint can have up to one link)
  - Position and rotation offsets w.r.t. the parent joint
  - Position, velocity, and torque (effort) limits
  - Other information (PID gains, damping factor, etc)

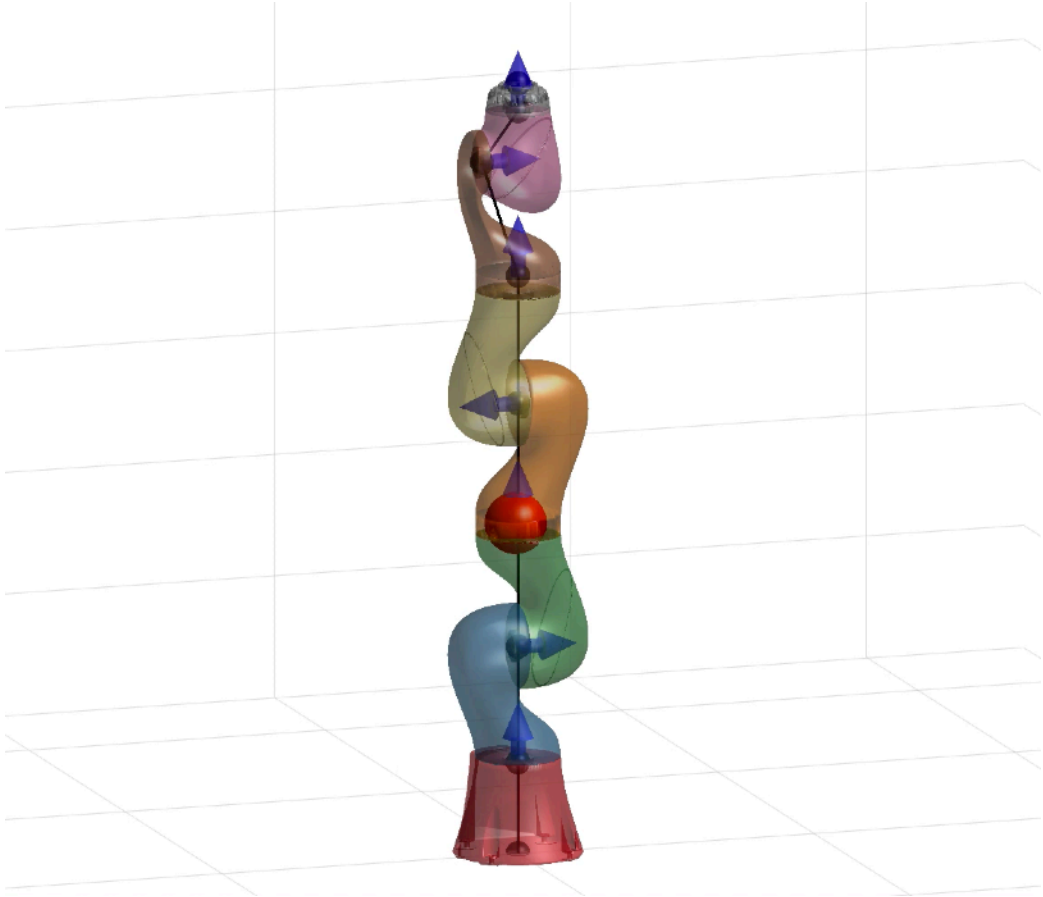
Link


```
<link name="iiwa7_link_1">
    <inertial>
        <origin xyz="0 -0.03 0.12" rpy="0 0 0"/>
        <mass value="3.4525"/>
        <inertia ixx="0.02183"  ixy="0"  ixz="0" iyy="0.007703" iyz="-0.003887" izz="0.02083" />
    </inertial>
    <visual>
        <origin xyz="0 0 0.0075" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://iiwa_description/meshes/iiwa7/visual/link_1.stl"/>
        </geometry>
        <material name="Orange"/>
    </visual>
    <collision>
        <origin xyz="0 0 0.0075" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://iiwa_description/meshes/iiwa7/collision/link_1.stl"/>
        </geometry>
        <material name="Orange"/>
    </collision>
</link>
```

- **Link** field contains
  - link name
  - Physical property (offset, mass, and moment of inertia)
  - Visual property (offset, file path of a CAD file (e.g., stl, dae, obj, color, etc)
  - Collision property (offset, file path of a CAD file mostly in a stl format

Parsing a URDF file

ROBOT INTELLIGENCE LAB

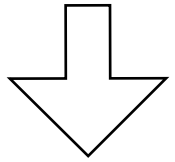


- We will parse KUKA LBR IIWA 7.

Parse an XML file

ROBOT INTELLIGENCE LAB

```
% Parse raw xml file
robot_name = 'iiwa7';
urdf_path = sprintf('../urdf/%s/%s_urdf.xml',robot_name,robot_name);
s = xml2str(urdf_path);
robot = s.robot;
fprintf('\n [%s] is parsed.\n\n',urdf_path);
disp(robot);
```



```
[../urdf/iiwa7/iiwa7_urdf.xml] is parsed.

        joint: {1×9 cell}
         link: {1×9 cell}
      Comment: 'joint between base and link_0 joint between link_0 and link_1 '
   Attributes: [1×1 struct]
```

Parse Joint Information

ROBOT INTELLIGENCE LAB

```
% Parse information
chain = init_chain('name',robot_name);
n_joint = length(robot.joint); % number of joints

% Basic joint and link information
joint_names       = cell(1,n_joint);
parent_link_names = cell(1,n_joint);
child_link_names  = cell(1,n_joint);
rev_joint_names = {}; n_rev_joint = 0;
for i_idx = 1:n_joint % for all joints
    joint_i = robot.joint{i_idx};
    joint_name = joint_i.Attributes.name;
    joint_type = joint_i.Attributes.type;
    joint_names{i_idx} = joint_name;
    parent_link_names{i_idx} = joint_i.parent.Attributes.link;
    child_link_names{i_idx} = joint_i.child.Attributes.link;
    if isequal(joint_type,'revolute')
        n_rev_joint = n_rev_joint + 1;
        rev_joint_names{n_rev_joint} = joint_name;
    end
end % for i_idx = 1:n_joint % for all joints
chain.joint_names     = joint_names;
chain.n_joint         = n_joint;
chain.rev_joint_names = rev_joint_names;
chain.n_rev_joint     = n_rev_joint;
```

Parse Joint Information

ROBOT INTELLIGENCE LAB

```
% Parse information
chain = init_chain('name',robot_name);
n_joint = length(robot.joint); % number of joints

% Basic joint and link information
joint_names       = cell(1,n_joint);
parent_link_names = cell(1,n_joint);
child_link_names  = cell(1,n_joint);
rev_joint_names = {}; n_rev_joint = 0;
for i_idx = 1:n_joint % for all joints
    joint_i = robot.joint{i_idx};
    joint_name = joint_i.Attributes.name;
    joint_type = joint_i.Attributes.type;
    joint_names{i_idx} = joint_name;
    parent_link_names{i_idx} = joint_i.parent.Attributes.link;
    child_link_names{i_idx} = joint_i.child.Attributes.link;
    if isequal(joint_type,'revolute')
        n_rev_joint = n_rev_joint + 1;
        rev_joint_names{n_rev_joint} = joint_name;
    end
end % for i_idx = 1:n_joint % for all joints
chain.joint_names     = joint_names;
chain.n_joint         = n_joint;
chain.rev_joint_names = rev_joint_names;
chain.n_rev_joint     = n_rev_joint;
```

```
function chain = init_chain(varargin)
%
% Initialize a kinematic chain
%

% Parse options
ps = inputParser;
addParameter(ps,'name','kinematic_chain');
addParameter(ps,'dt',0.01);
parse(ps,varargin{:});
dt      = ps.Results.dt;
name    = ps.Results.name;

% Initialize a kinematic chain
chain                   = struct();
chain.name              = name;
chain.dt                = dt;
chain.joint             = struct(...
    'name','',...                   % joint name
    'p',cv([0,0,0]),...             % joint position
    'R',eye(3,3),...                % joint orientation
    'a',cv([0,0,0]),....            % rotation axis
    'type','fixed',...              % rotation type
    'p_offset',cv([0,0,0]),...      % position offset (w.r.t. parent joint)
    'R_offset',eye(3,3),...         % rotation offset (w.r.t. parent joint)
    'q',0,...                       % joint position
    'dq',0,...                      % dq/dt
    'ddq',0,...                     % d^2q/dt^2
    'q_diff',0,...                  % joint position different (q_diff/dt = dq)
    'q_prev',0,...                  % previous joint position
    'v',cv([0,0,0]),...             % linear velocity in {W}
    'vo',cv([0,0,0]),...            % linear part of spatial velocity
    'w',cv([0,0,0]),...             % angular velocity
    'dvo',cv([0,0,0]),...           % linear spatial acceleration
    'dw',cv([0,0,0]),...            % angular acceleration
    'u',0,...                       % joint torque
    'ext_f',cv([0,0,0]),...         % external force
    'parent',[],...                 % parent joint
    'childs',[],...                 % child joint
    'link_idx',[],...               % connected link index
    'limit',[-inf,+inf]...          % joint limit
    );
chain.joint_names       = {};
chain.n_joint           = 0;
chain.rev_joint_names   = {};
chain.n_rev_joint       = 0;
```

Parse Joint Information

ROBOT INTELLIGENCE LAB

```
% Parse information
chain = init_chain('name',robot_name);
n_joint = length(robot.joint); % number of joints

% Basic joint and link information
joint_names       = cell(1,n_joint);
parent_link_names = cell(1,n_joint);
child_link_names  = cell(1,n_joint);
rev_joint_names = {}; n_rev_joint = 0;
for i_idx = 1:n_joint % for all joints
    joint_i = robot.joint{i_idx};
    joint_name = joint_i.Attributes.name;
    joint_type = joint_i.Attributes.type;
    joint_names{i_idx} = joint_name;
    parent_link_names{i_idx} = joint_i.parent.Attributes.link;
    child_link_names{i_idx} = joint_i.child.Attributes.link;
    if isequal(joint_type,'revolute')
        n_rev_joint = n_rev_joint + 1;
        rev_joint_names{n_rev_joint} = joint_name;
    end
end % for i_idx = 1:n_joint % for all joints
chain.joint_names     = joint_names;
chain.n_joint         = n_joint;
chain.rev_joint_names = rev_joint_names;
chain.n_rev_joint     = n_rev_joint;
```

First, parse joint names and connected link names to construct the kinematic chain (parent-child structure)
+ accumulate revolute joints

```
%% Parse detailed joint information
for i_idx = 1:n_joint % for all joints
    joint_i = robot.joint{i_idx};
    joint_name = joint_i.Attributes.name;
    joint_type = joint_i.Attributes.type;
    % Joint position limit
    if isfield(joint_i,'limit')
        limit_lower = joint_i.limit.Attributes.lower;
        upper = joint_i.limit.Attributes.upper;
        if limit_lower(1) == '$', limit_lower = eval(limit_lower(3:end-1));
        else, limit_lower = eval(limit_lower);
        end
        if upper(1) == '$', upper = eval(upper(3:end-1));
        else, upper = eval(upper);
        end
        limit = [limit_lower,upper];
    else
        if isequal(joint_type,'revolute')
            limit = [-inf,+inf]; % revolute joint without limit specified
        else
            limit = [0,0]; % default fixed joint limit
        end
    end
    % Parent joint
    parent_joint_idx = idx_cell(child_link_names,parent_link_names{i_idx});
    if isempty(parent_joint_idx)
        parent_joint_name = '';
    else
        parent_joint_name = joint_names{parent_joint_idx};
    end
    % position and rotation offset
    if isfield(joint_i,'axis')
        a = cv(str2num(joint_i.axis.Attributes.xyz));
    else
        a = cv([0,0,0]);
    end
    % Joint position offset
    if isfield(joint_i,'origin')
        p_offset = cv(str2num(joint_i.origin.Attributes.xyz));
        R_offset = rpy2r(str2num(joint_i.origin.Attributes.rpy));
    else
        p_offset = cv([0,0,0]);
        R_offset = eye(3,3);
    end
    % Add joint
    q = 0; dq = 0; ddq = 0;
    chain.joint(i_idx) = struct(...
        'name',joint_name,...
        'p',cv([0,0,0]),'R',eye(3,3),'a',cv(a),'type',joint_type,...
        'p_offset',cv(p_offset),'R_offset',R_offset,...
        'q',q,'dq',dq,'ddq',ddq,'q_diff',0,'q_prev',0,...
        'v',cv([0,0,0]),'vo',cv([0,0,0]),'w',cv([0,0,0]),...
        'dvo',cv([0,0,0]),'dw',cv([0,0,0]),'u',0,'ext_f',cv([0,0,0]),...
        'parent',[],'childs',[],'link_idx',[],'limit',limit...
        );
    % Add parent joint index
    chain.joint(i_idx).parent = parent_joint_idx;
end % for i_idx = 1:n_joint % for all joints
```

Joint names and types

Joint position limit

Parent joint name

Joint axis

Position and rotation offset

Add joint information

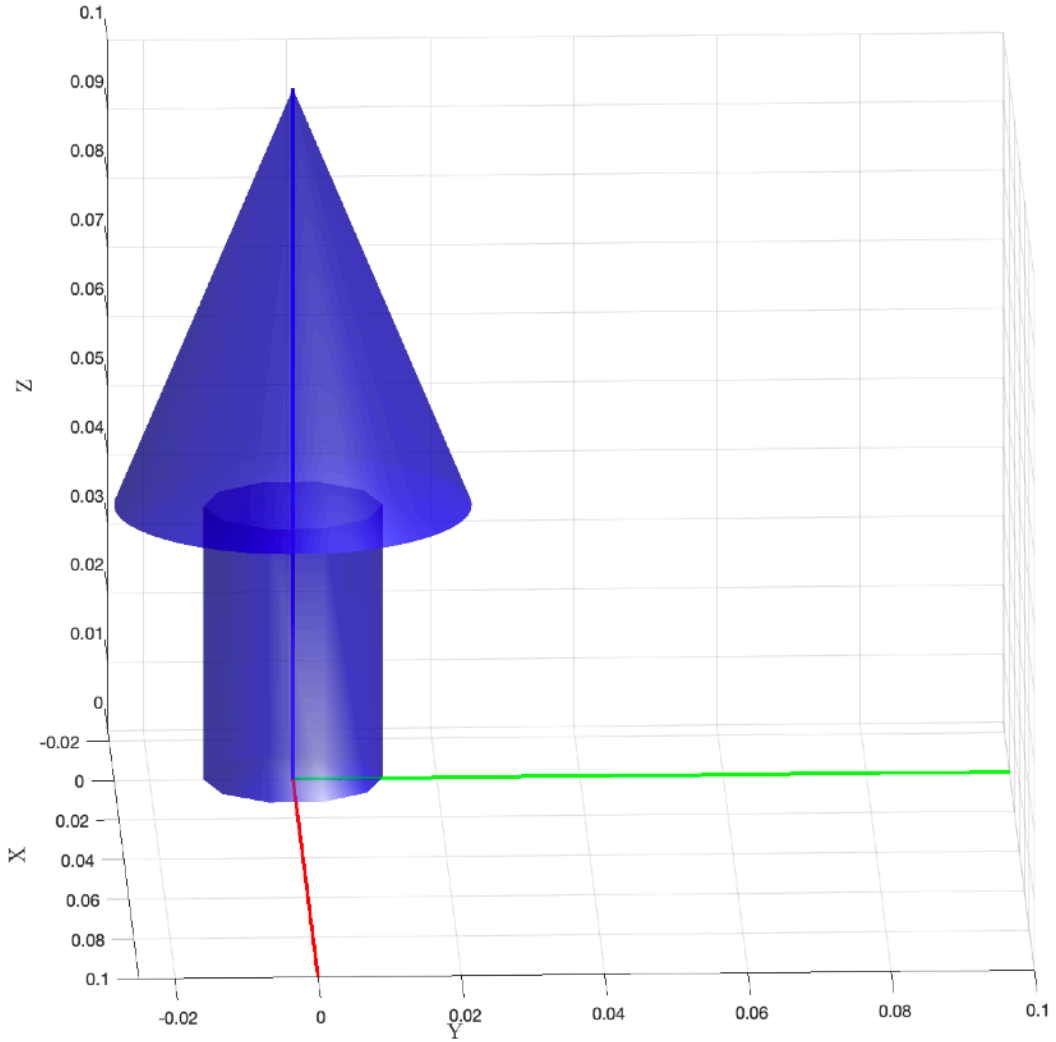
ROBOT INTELLIGENCE LAB

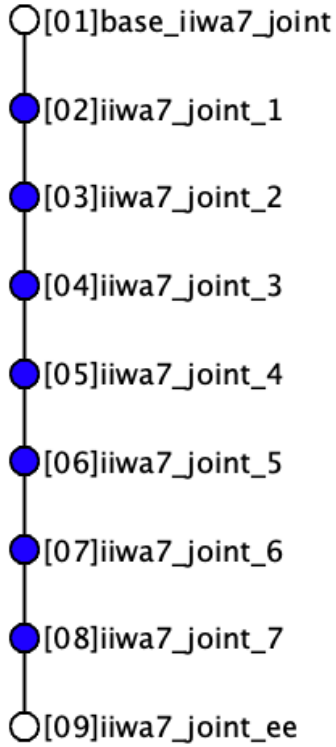
ROBOT INTELLIGENCE LAB

```
%% Add the current joint as childs of the parent joint
for i_idx = 1:n_joint % for all joints
    joint_i = chain.joint(i_idx);
    parent_idx = joint_i.parent;
    if ~isempty(parent_idx)
        childs = chain.joint(parent_idx).childs;
        childs = [childs, i_idx]; % append
        chain.joint(parent_idx).childs = childs;
    end
end
```

Append children

Plot?

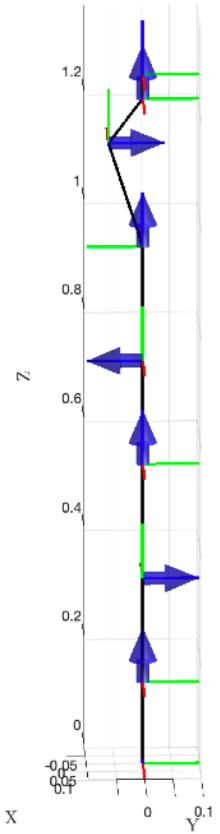

0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
Z
-0.02
0
0.02
0.04
0.06
0.08
0.1
X
-0.02
0
0.02
0.04
0.06
0.08
0.1
Y


[01]base_iiwa7_joint
[02]iiwa7_joint_1
[03]iiwa7_joint_2
[04]iiwa7_joint_3
[05]iiwa7_joint_4
[06]iiwa7_joint_5
[07]iiwa7_joint_6
[08]iiwa7_joint_7
[09]iiwa7_joint_ee

Forward Kinematics

ROBOT INTELLIGENCE LAB

```
%% Forward kinematics then plot
ca;
chain = fk_chain(chain,'');
plot_chain(chain,'PLOT_JOINT_AXIS',1)
plot_chain_graph(chain)
```



Parse Link

ROBOT INTELLIGENCE LAB

```
%% Parse link information
n_link = length(robot.link); % number of link
chain.n_link = n_link;
chain.link_names = cell(1,n_link);
for i_idx = 1:chain.n_link % for all links
    link_i = robot.link{i_idx};
    link_name = link_i.Attributes.name;
    chain.link_names{i_idx} = link_name; % append link names
end % for i_idx = 1:chain.n_link % for all links
```

```
for i_idx = 1:chain.n_link % for all links
    link_i = robot.link{i_idx};
    link_name = link_i.Attributes.name;
    joint_idx = idx_cell(child_link_names,link_name); % attached joint index
    % Parse link mesh offset
    p_offset = cv([0,0,0]); R_offset = eye(3,3);
    if isfield(link_i,'visual')
        if isfield(link_i.visual,'origin')
            xyz = cv(str2num(link_i.visual.origin.Attributes.xyz));
            rpy = cv(str2num(link_i.visual.origin.Attributes.rpy));
            p_offset = xyz;
            R_offset = rpy2r(rpy);
        end
    end
    % Parse link mesh name and scale (if exist)
    filename = ''; mesh_path = ''; scale = cv([1,1,1]);
    if isfield(link_i,'visual')
        if isfield(link_i.visual,'geometry')
            if isfield(link_i.visual.geometry,'mesh')
                if isfield(link_i.visual.geometry.mesh.Attributes,'filename')
                    filename = link_i.visual.geometry.mesh.Attributes.filename;
                end
                if isfield(link_i.visual.geometry.mesh.Attributes,'scale')
                    scale = cv(str2num(link_i.visual.geometry.mesh.Attributes.scale));
                end
            end
        end
    end
    if ~isempty(filename) % if file name exists
        [~,name,ext] = fileparts(filename);
        [f,~,~] = fileparts(urdf_path);
        mesh_path = [f,'/visual/',name,ext];
    end
    if (~isempty(mesh_path)) && (~exist(mesh_path,'file'))
        % Check whether the specified stl file exist or not
        fprintf(2,'[%s] does not exist.\n',mesh_path);
    end
    % Load stl file
    fv = '';
    if exist(mesh_path,'file') % if file exists
        [~,~,ext] = fileparts(mesh_path);
        switch lower(ext)
            case '.stl'
                fv = load_stl(mesh_path);                    % load stl
                [fv.vertices,fv.faces]= patchslim(fv.vertices,fv.faces);    % reduce mesh
                fv.vertices = rv(scale).*fv.vertices;        % scaling
                fv.vertices = fv.vertices * R_offset';       % rotate mesh
                fv.vertices = fv.vertices + p_offset';       % translate mesh
            otherwise
                fprintf(2,'Unsupported file type:[%s]. Only .stl is supported. \n',ext);
        end
    end % if exist(chain.link(i_idx).mesh_path,'file') % if file exists
```

Attached Joint

Link Offset

Link Mesh Information

ROBOT INTELLIGENCE LAB

ROBOT INTELLIGENCE LAB

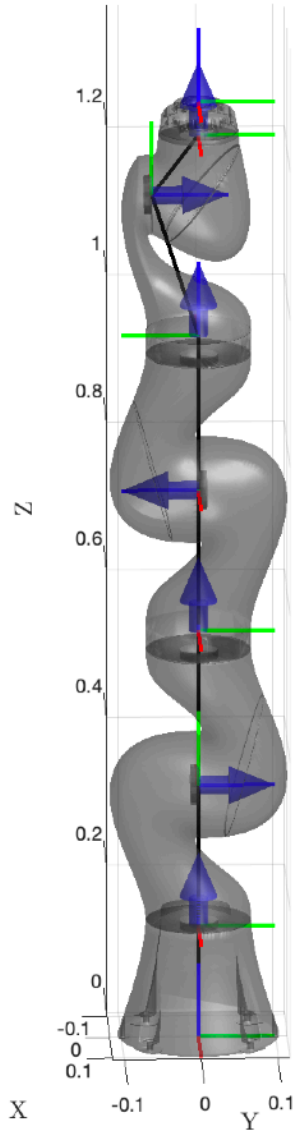
```
        % Parse box information
        box_size = ''; box_scale = cv([1,1,1]); box = '';
        if isfield(link_i,'visual')
            if isfield(link_i.visual,'geometry')
                if isfield(link_i.visual.geometry,'box')
                    if isfield(link_i.visual.geometry.box.Attributes,'size')
                        box_size = cv(str2num(link_i.visual.geometry.box.Attributes.size));
                    end
                    if isfield(link_i.visual.geometry.box.Attributes,'scale')
                        box_scale = cv(str2num(link_i.visual.geometry.box.Attributes.scale));
                    end
                    if ~isempty(box_size)
                        box = struct('size',box_size,'scale',box_scale);
                    end
                end
            end
        end
        % Append chain link
        chain.link(i_idx).name      = link_name;
        chain.link(i_idx).joint_idx = joint_idx;
        chain.link(i_idx).p_offset  = p_offset;
        chain.link(i_idx).R_offset  = R_offset;
        chain.link(i_idx).mesh_path = mesh_path;
        chain.link(i_idx).scale     = scale;
        chain.link(i_idx).fv        = fv; % append mesh
        chain.link(i_idx).box       = box;
    end % for i_idx = 1:chain.n_link % for all links
```

Box Information

Append Link

Now, we have meshes

ROBOT INTELLIGENCE LAB



Optimize Capsule

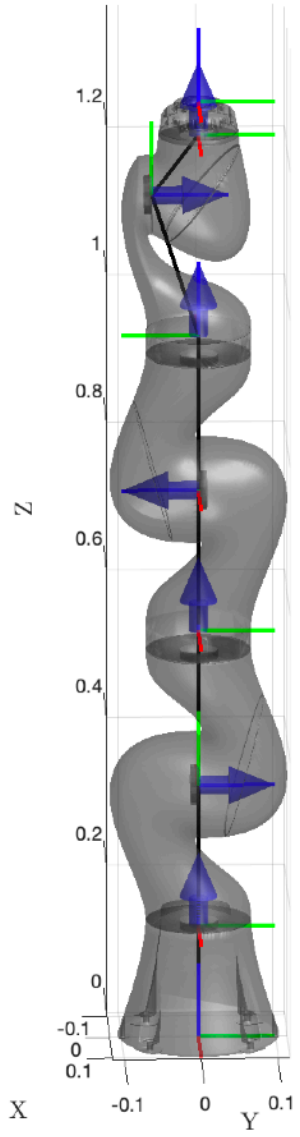
ROBOT INTELLIGENCE LAB

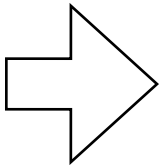
```
%% Optimize capsules
ca
SKIP_CAPSULE = 0;
for i_idx = 1:chain.n_link % for each link
    link_i = chain.link(i_idx);
    fv_i = link_i.fv;
    if (~isempty(fv_i)) && (~SKIP_CAPSULE)
        cap_opt_i = optimize_capsule(fv_i); % optimize capsule (takes time)
    else
        cap_opt_i = '';
    end
    chain.link(i_idx).capsule = cap_opt_i;
end % for i_idx = 1:chain.n_link % for each link
fprintf('Done.\n');
```

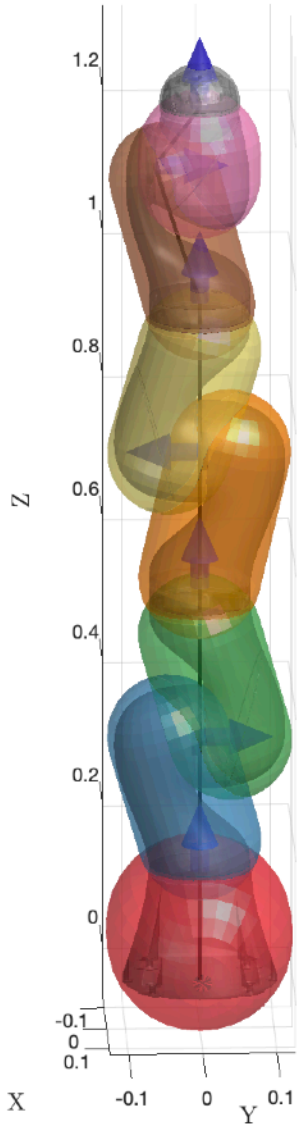
```
function cap_opt = optimize_capsule(fv)
%
% Optimize capsule
%

% Reduce once more
reduce_patch_ratio = min(1,600/size(fv.vertices,1)); % #vertex is 100
fv = reducepatch(fv,reduce_patch_ratio);
% Initial guess
radius = max(max(fv.vertices)-min(fv.vertices));
height = max(max(fv.vertices)-min(fv.vertices));
% Optimize
res = 30;
pdata = fv.vertices;
mean_fv = mean(fv.vertices);
use_vector_operation = 1;
x0 = [mean_fv(1) mean_fv(2) -1.5*height ...
    mean_fv(1) mean_fv(2) 1.5*height ...
    radius*2]'; % initial guess
algorithm = 'interior-point';  % 'sqp' 'interior-point'
options = optimoptions('fmincon','Algorithm',algorithm,'display','off',...
    'StepTolerance',1e-10,'OptimalityTolerance',1e-10);
A = []; B = []; Aeq = []; Beq = []; LB = []; UB = [];
NONLCON = @(x)mycon(x,pdata,use_vector_operation);
[x,fval] = fmincon(@(x)myfun(x),x0,A,B,Aeq,Beq,LB,UB,NONLCON,options);
e1 = x(1:3); e2 = x(4:6); r = x(7);
height_opt = norm(e2-e1);
radius_opt = r;
p_opt = (e1 + e2)/2;
diff_vec = e2-e1; diff_unit = diff_vec / norm(diff_vec,'fro');
R_opt = vecRotMat([0 0 1], diff_unit');
cap_opt = get_capsule_shape(...
    'T_offset',pr2t(p_opt,R_opt),'radius',radius_opt,'height',height_opt,'res',res);
cap_opt.p = p_opt;
cap_opt.R = R_opt;
cap_opt.height = height_opt;
cap_opt.radius = radius_opt;
cap_opt.fval = fval;
end
```

Now, we have capsules

ROBOT INTELLIGENCE LAB







Why capsules?

ROBOT INTELLIGENCE LAB

The distance between two capsules can be easily computed.

Check Self-Collision

ROBOT INTELLIGENCE LAB

```
%% Check self-collision
clc;ca;

% Get link indices to check self-collision
chain.sc_checks = get_sc_checks(chain,'collision_margin',max(chain.sz.xyz_len)/100);

% Check self-collision
while 1
    % Random robot pose
    q_rand = zeros(1,chain.n_rev_joint);
    for i_idx = 1:chain.n_rev_joint
        limit_i = chain.joint(idx_cell(chain.joint_names,chain.rev_joint_names
        q_rand(i_idx) = limit_i(1) + (limit_i(2)-limit_i(1))*rand;
    end
    chain = update_chain_q(chain,chain.rev_joint_names,q_rand);
    [SC,sc_link_pairs] = check_sc(chain,'collision_margin',0);
    if SC
        break;
    end
end

% Plot
plot_chain(chain,'fig_idx',1,'subfig_idx',1,'fig_pos',[0.0,0.5,0.4,0.5],...
    'PLOT_BOX',1,'bec','k');
for sc_idx = 1:size(sc_link_pairs,1)
    sc_link_pair = sc_link_pairs(sc_idx,:);
    i_idx = sc_link_pair(1); j_idx = sc_link_pair(2);
    % Link i
    [cap_i,T_i] = get_chain_capsule(chain,i_idx);
    % Link j
    [cap_j,T_j] = get_chain_capsule(chain,j_idx);
    % Plot colliding capsules
    plot_capsule(cap_i,'fig_idx',1,'subfig_idx',2*sc_idx-1,...
        'T',T_i,'cfc','r','cfa',0.5,'cec','k','cea',0.5);
    plot_capsule(cap_j,'fig_idx',1,'subfig_idx',2*sc_idx,...
        'T',T_j,'cfc','r','cfa',0.5,'cec','k','cea',0.5);
end
if SC
    title_str = sprintf('Self Collision Occurred');
    plot_title(title_str,'tfc','r','tfs',20);
else
    title_str = sprintf('No Collision');
    plot_title(title_str,'tfc','k','tfs',20);
end
fprintf('Done.\n');
```

Get link pairs to check self-collision

```
function sc_checks = get_sc_checks(chain,varargin)
%
% Get link indices to check self-collision
%

% Parse options
ps = inputParser;
addParameter(ps,'collision_margin',max(chain.sz.xyz_len)/100);
parse(ps,varargin{:});
collision_margin = ps.Results.collision_margin;

sc_checks = [];
for i_idx = 1:chain.n_link % for all links
    link_i = chain.link(i_idx);
    cap_i = link_i.capsule;
    if isempty(cap_i), continue; end % ignore empty capsule
    joint_idx_i = link_i.joint_idx;
    if isempty(joint_idx_i)
        joint_idx_i = get_topmost_idx(chain);
    end
    T_i = pr2t(chain.joint(joint_idx_i).p,chain.joint(joint_idx_i).R);
    cl_i = get_capsule_line(T_i,cap_i);
    for j_idx = i_idx+1:chain.n_link
        link_j = chain.link(j_idx);
        cap_j = link_j.capsule;
        if isempty(cap_j), continue; end % ignore empty capsule
        joint_idx_j = link_j.joint_idx;
        if isempty(joint_idx_j)
            joint_idx_j = get_topmost_idx(chain);
        end
        T_j = pr2t(chain.joint(joint_idx_j).p,chain.joint(joint_idx_j).R);
        cl_j = get_capsule_line(T_j,cap_j);
        line_dist = get_dist_lines(cl_i.p1,cl_i.p2,cl_j.p1,cl_j.p2);
        cap_dist = line_dist - cap_i.radius - cap_j.radius;
        if cap_dist > collision_margin
            sc_checks = cat(1,sc_checks,[i_idx,j_idx]);
        end
    end
end
```

Check Self-Collision

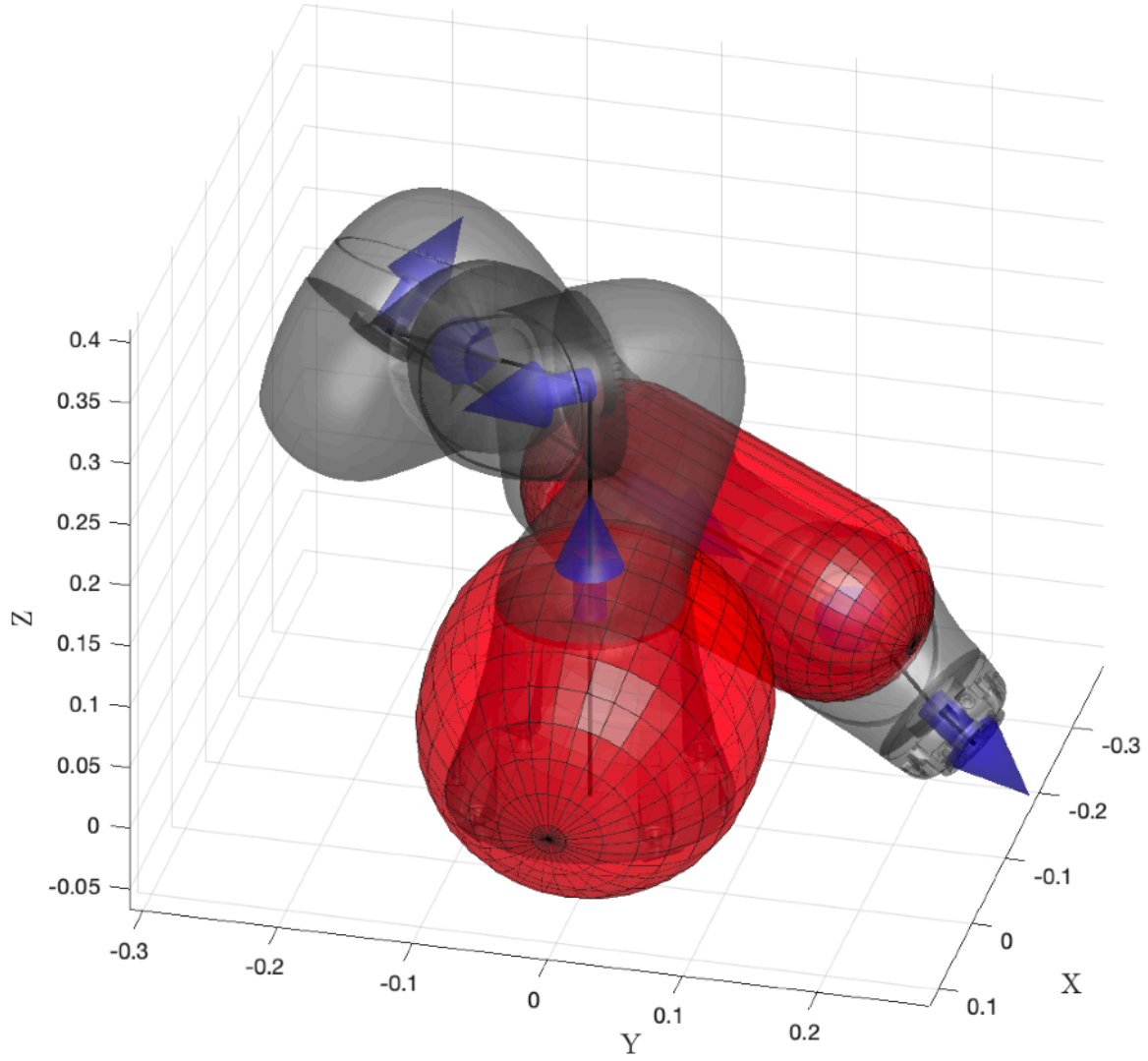
ROBOT INTELLIGENCE LAB

```
%% Check self-collision
clc;ca;

% Get link indices to check self-collision
chain.sc_checks = get_sc_checks(chain,'collision_margin',max(chain.sz.xyz_len)/100);

% Check self-collision
while 1
    % Random robot pose
    q_rand = zeros(1,chain.n_rev_joint);
    for i_idx = 1:chain.n_rev_joint
        limit_i = chain.joint(idx_cell(chain.joint_names,chain.rev_joint_names{i_idx})).limit;
        q_rand(i_idx) = limit_i(1) + (limit_i(2)-limit_i(1))*rand;
    end
    chain = update_chain_q(chain,chain.rev_joint_names,q_rand);
    [SC,sc_link_pairs] = check_sc(chain,'collision_margin',0);
    if SC
        break;
    end
end

% Plot
plot_chain(chain,'fig_idx',1,'subfig_idx',1,'fig_pos',[0.0,0.5,0.4,0.5],...
    'PLOT_BOX',1,'bec','k');
for sc_idx = 1:size(sc_link_pairs,1)
    sc_link_pair = sc_link_pairs(sc_idx,:);
    i_idx = sc_link_pair(1); j_idx = sc_link_pair(2);
    % Link i
    [cap_i,T_i] = get_chain_capsule(chain,i_idx);
    % Link j
    [cap_j,T_j] = get_chain_capsule(chain,j_idx);
    % Plot colliding capsules
    plot_capsule(cap_i,'fig_idx',1,'subfig_idx',2*sc_idx-1,...
        'T',T_i,'cfc','r','cfa',0.5,'cec','k','cea',0.5);
    plot_capsule(cap_j,'fig_idx',1,'subfig_idx',2*sc_idx,...
        'T',T_j,'cfc','r','cfa',0.5,'cec','k','cea',0.5);
end
if SC
    title_str = sprintf('Self Collision Occurred');
    plot_title(title_str,'tfc','r','tfs',20);
else
    title_str = sprintf('No Collision');
    plot_title(title_str,'tfc','k','tfs',20);
end
fprintf('Done.\n');
```

Loop until we have self-collision

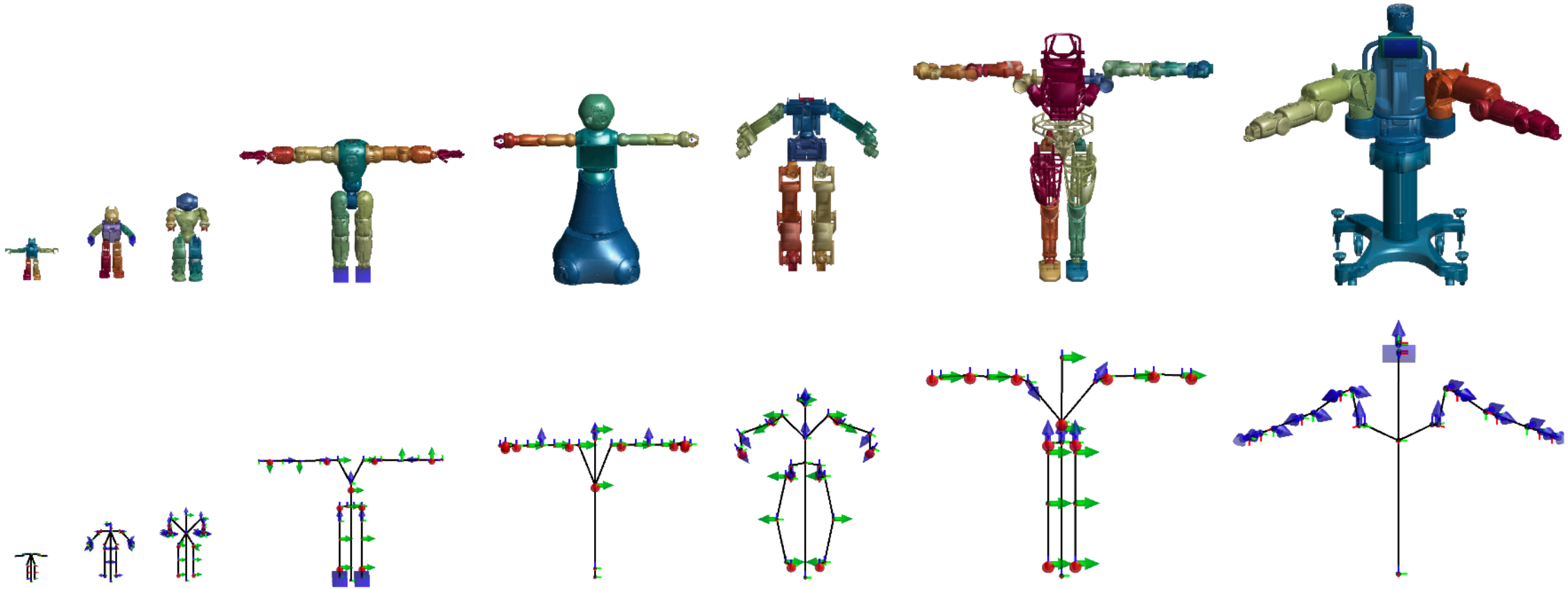
Check Self-Collision

ROBOT INTELLIGENCE LAB



More Robots!

ROBOT INTELLIGENCE LAB



Thank You