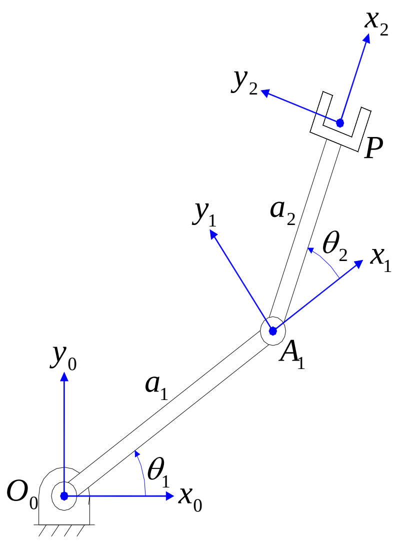
CH. 2: SimMechanics Simulation Practice

[1] Forward Dynamics Simulation Practice of a 2-DOF Planar Robot



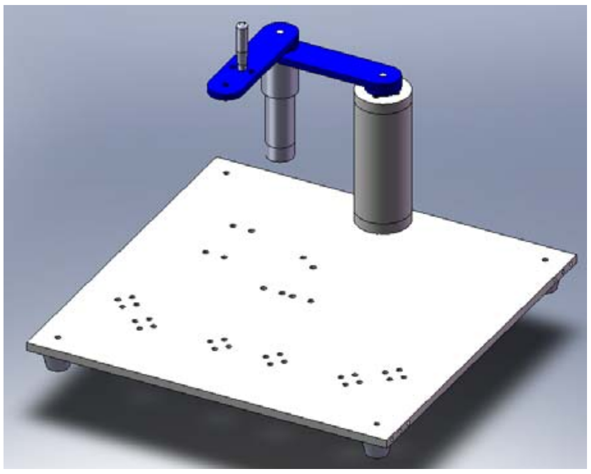


Fig. 1: Planar 2-DOF serial robot

(1) Parameters

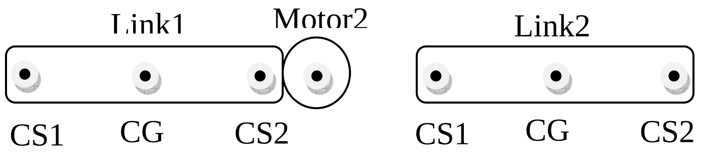


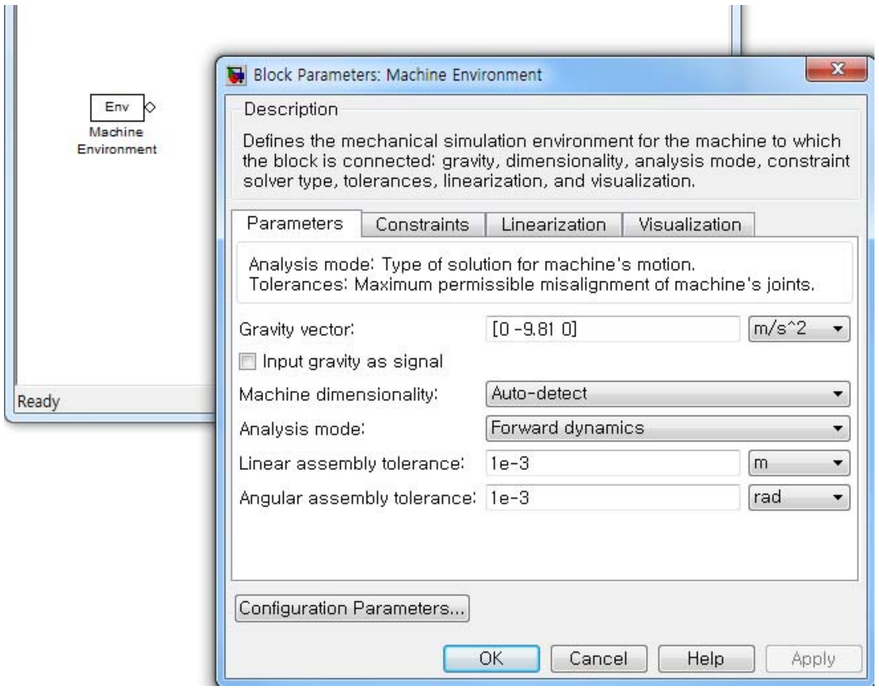
Table 1: Mass properties and joint locations

	Mass(kg)	Inertia(kg mm^2)	CG Position(m) /from CS1	CS(m) /from World
Ground	-	-	-	[0 0 0]
Link1	0.071	89.395	[0.0373 0 0]	CS1 = [0 0 0] CS2 = [0.1 0 0]
Motor2	0.292	0	[0 0 0]	CS1 = [0.1 0 0] CS2 = [0.1 0 0]
Link2	0.075	109.864	[0.04168 0 0]	CS1 = [0.1 0 0] CS2 = [0.2 0 0]

- Link lengths: $a_1 = 100$ mm , $a_2 = 100$ mm
- Here, CG and CS denote the center of gravity and coordinate system of each joint location.

(2) Modeling method

- First, click Simulink Library Browser → Simscape → SimMechanics First Generation → Bodies library.
- Create new model by File → New → Model.
- Drag Machine Environment in Bodies into the model.



Description

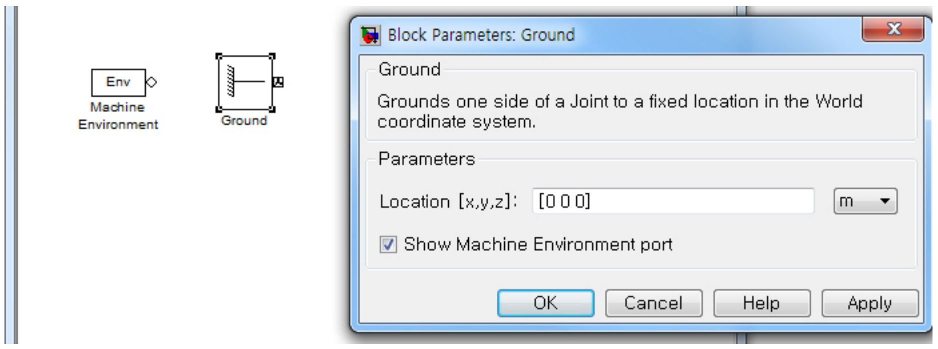
Defines the mechanical simulation environment for the machine to which the block is connected: gravity, dimensionality, analysis mode, constraint solver type, tolerances, linearization, and visualization.

Analysis mode: Type of solution for machine's motion.
Tolerances: Maximum permissible misalignment of machine's joints.

Gravity vector: [0 -9.81 0] m/s^2
Input gravity as signal
Machine dimensionality: Auto-detect
Analysis mode: Forward dynamics
Linear assembly tolerance: 1e-3 m
Angular assembly tolerance: 1e-3 rad

Fig. 2: Machine Environment block

- Machine Environment contains basic settings.
- Note that the gravity direction be set to –y direction [0 -9.81 0].
- Next, drag Ground block in Bodies Library.



Ground

Grounds one side of a Joint to a fixed location in the World coordinate system.

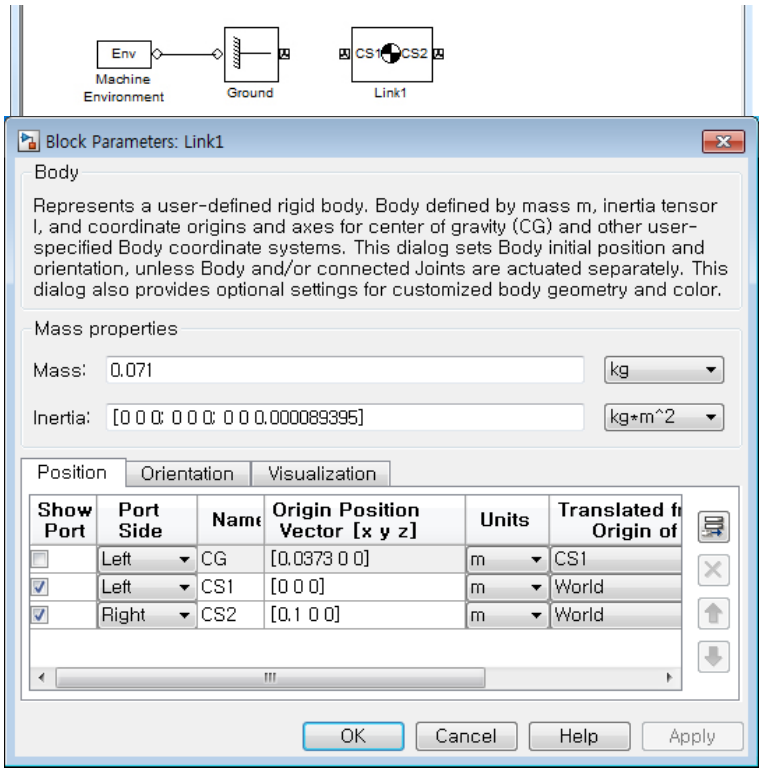
Parameters

Location [x,y,z]: [0 0 0] m

☑ Show Machine Environment port

Fig. 3: Ground block

- In Table 1, input [0 0 0] in Location parameter, and check “Show Machine Environment Port” so that the Ground block can be connected to other blocks.
- Next, drag Body block in Bodies.



Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

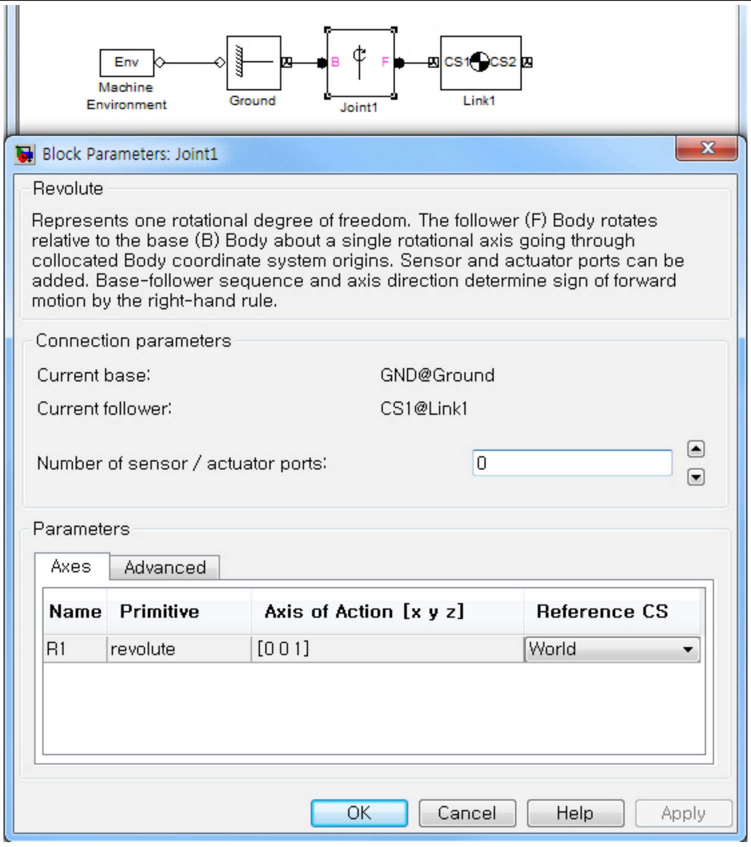
Mass: 0.071 kg

Inertia: [0 0 0; 0 0 0; 0 0 0.000089395] kg*m^2

Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated fr Origin of
☐	Left	CG	[0.0373 0 0]	m	CS1
✓	Left	CS1	[0 0 0]	m	World
✓	Right	CS2	[0.1 0 0]	m	World

Fig. 4: Link1 block

- Input the mass, inertia, and position vector parameters in Table 1.
- Since the link lies on the XY plane, the 3x3 inertia matrix may have only the Z element.
- Note units when inputting data.
- “Translated from Origin of” and “Components in Axes of” parameters denote the reference frames for both ends of link and CG point. It is convenient to express CG with respect to the CS1 frame and others in the World frame.
- Drag Revolute block in SimMechanics First Generation → Joints.
- Connect Revolute block to Ground and Link1 as shown in the following figure.



Revolute

Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

Connection parameters

Current base:	GND@Ground
Current follower:	CS1@Link1
Number of sensor / actuator ports:	0

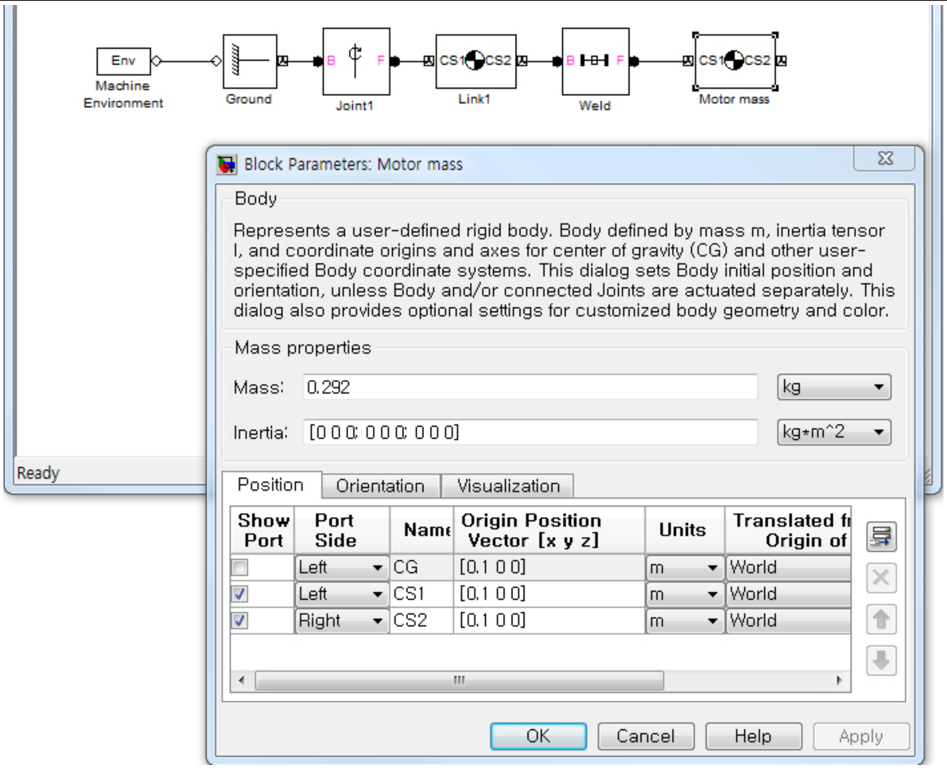
Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World

Fig. 5: Joint1 block

- Since links lie on the XY plane and the joint direction is the Z axis, set the “Axis of Action to the Z axis or [0 0 1].
- Similarly to modeling of Link1, make the modeling of the Motor mass with a Body block.
- Use “Weld block in Joints library to connect Motor to Link1.



Block Parameters: Motor mass

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass: 0.292 kg

Inertia: [0 0 0; 0 0 0; 0 0 0] kg*m^2

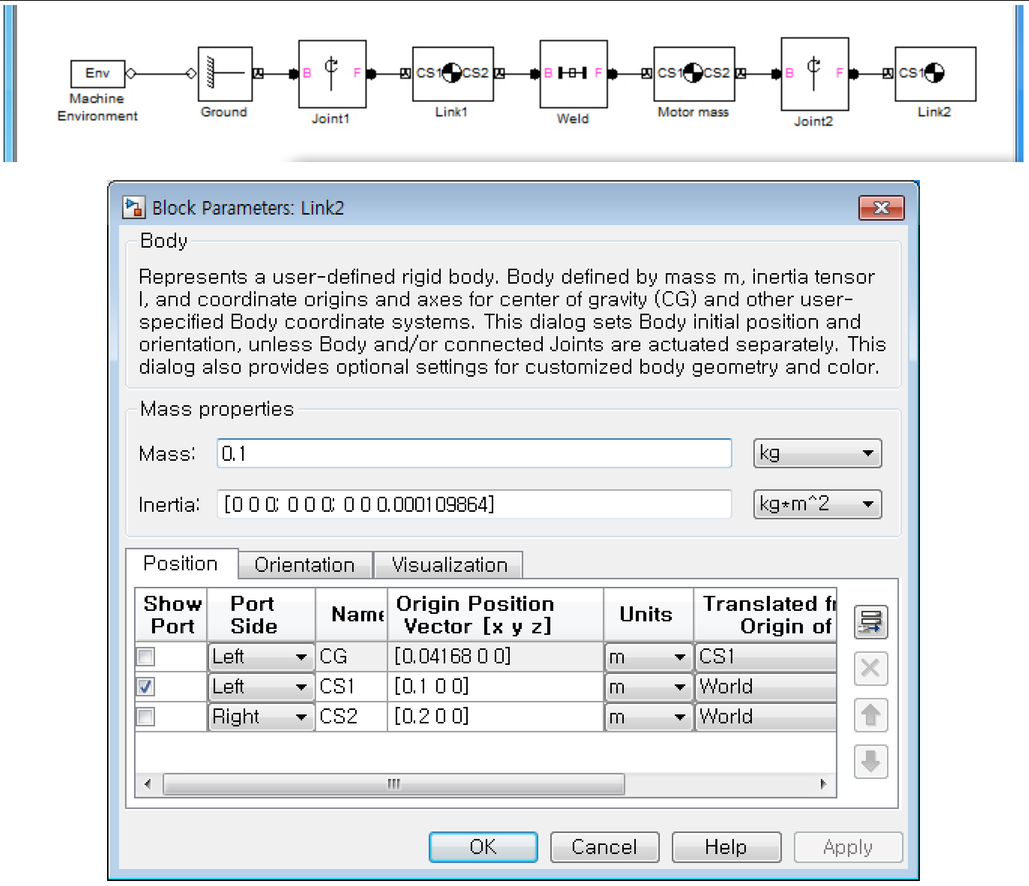
Position | Orientation | Visualization

Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated fr Origin of
☐	Left	CG	[0.1 0 0]	m	World
☑	Left	CS1	[0.1 0 0]	m	World
☑	Right	CS2	[0.1 0 0]	m	World

OK Cancel Help Apply

Fig. 6: Motor mass & Weld Joint block

- Insert Link2 and Joint2 into the model.
- Since the right end of Link2 is not connected to other blocks, uncheck “Right Port Side”.



Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass: 0.1 kg

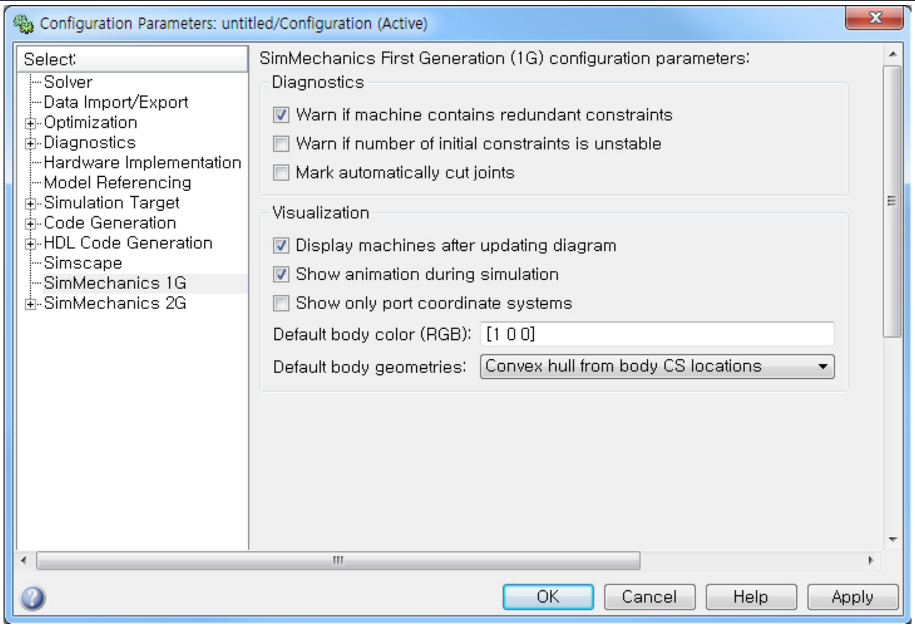
Inertia: [0 0 0; 0 0 0; 0 0 0.000109864] kg*m^2

Position | Orientation | Visualization

Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated fr Origin of
☐	Left	CG	[0.04168 0 0]	m	CS1
✓	Left	CS1	[0.1 0 0]	m	World
☐	Right	CS2	[0.2 0 0]	m	World

Fig. 7: Link2 & Joint2 block

- In order to check the model, let’s perform the animation.
- In the pull-down menu, select Simulation → Model Configuration Parameters.
- In the Configuration Parameters window, select SimMechanics 1G and check “Display machines after updating diagram” and “Show animation during simulation” in the Visualization section.



Configuration Parameters: untitled/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- HDL Code Generation
- Simscape
- SimMechanics 1G
- SimMechanics 2G

SimMechanics First Generation (1G) configuration parameters:

Diagnostics
- [✓] Warn if machine contains redundant constraints
- [ ] Warn if number of initial constraints is unstable
- [ ] Mark automatically cut joints

Visualization
- [✓] Display machines after updating diagram
- [✓] Show animation during simulation
- [ ] Show only port coordinate systems

Default body color (RGB): [1 0 0]

Default body geometries: Convex hull from body CS locations

OK Cancel Help Apply

Fig. 8: Configuration Parameters

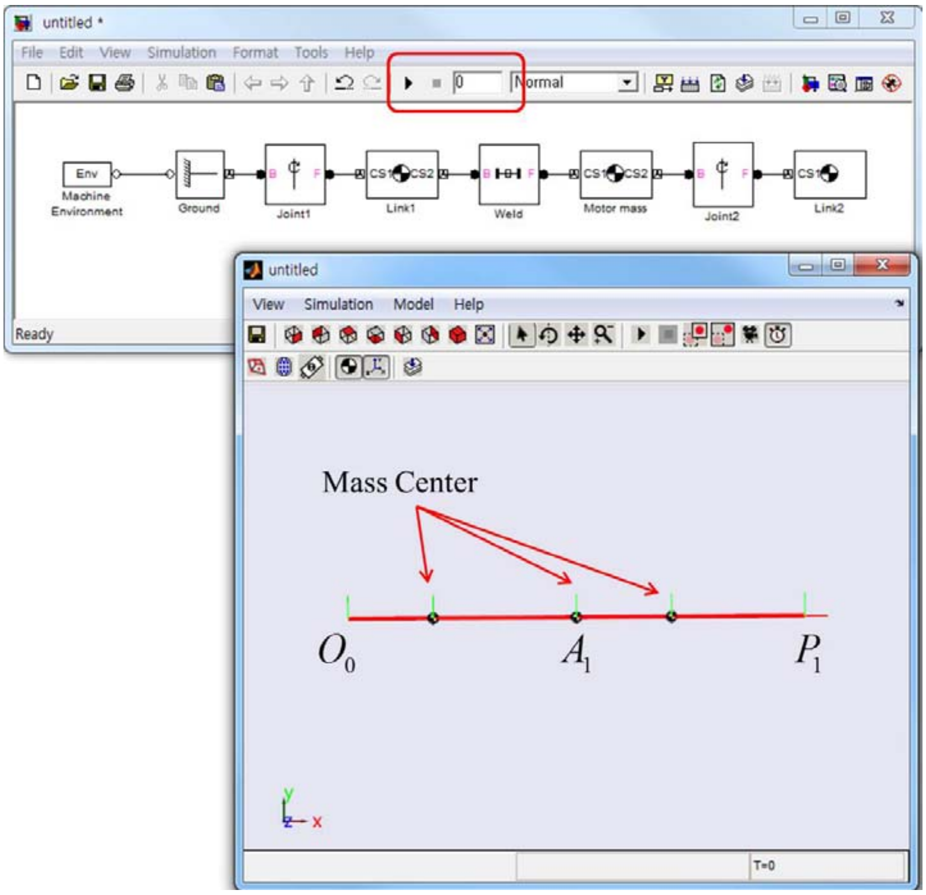
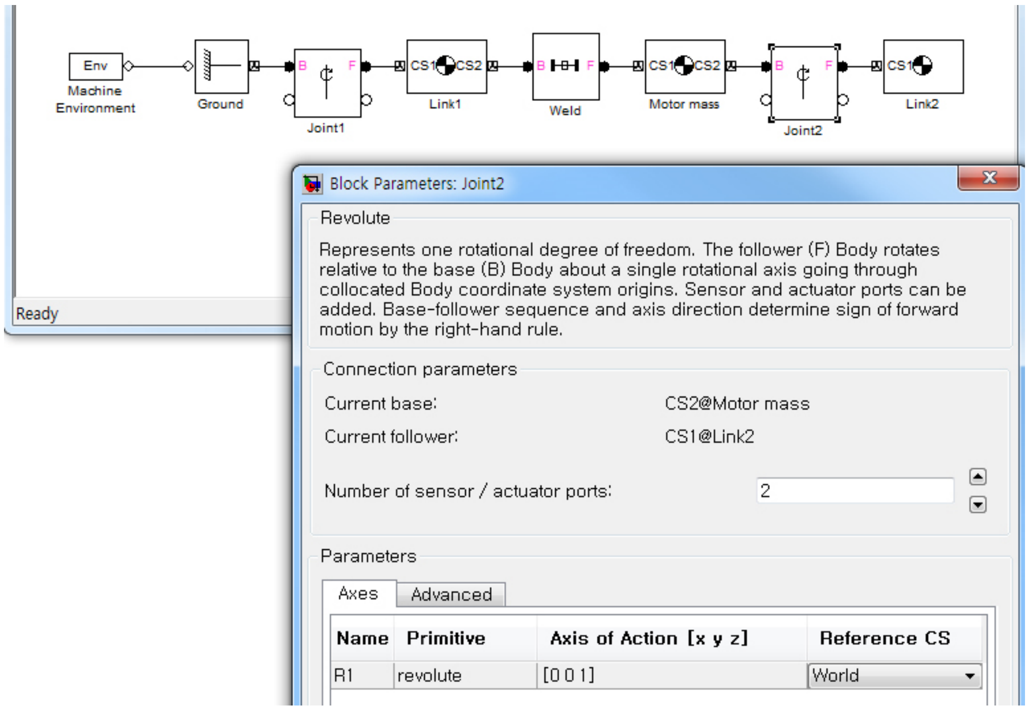


Fig. 9: Visualization of Model

- In order to check the initial position, set Simulation Stop Time to 0 and press the run icon (▶).

(3) Forward dynamics simulation

- This section explains the forward dynamics simulation of the 2-DOF serial robot or Double Pendulum.
- First, insert sensor/actuator ports in the two Joint blocks.



Revolute

Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

Connection parameters

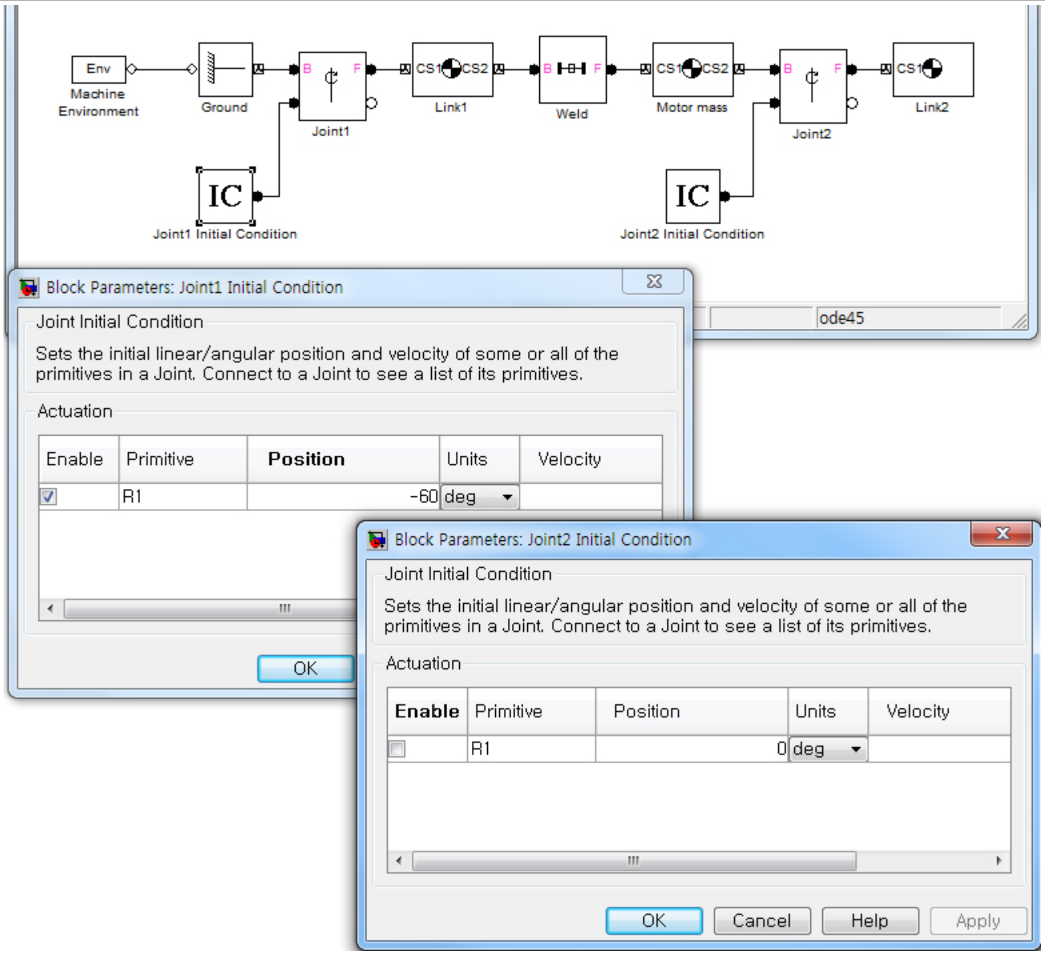
Current base:	CS2@Motor mass
Current follower:	CS1@Link2
Number of sensor / actuator ports:	2

Parameters

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World

Fig. 10: Add sensor/actuator ports

- Drag Joint Initial Condition block in Sensors & Actuators library into the model.
- Set the initial condition of Joint1 to -60° for Forward Dynamics Simulation.



Block Parameters: Joint1 Initial Condition

Joint Initial Condition

Sets the initial linear/angular position and velocity of some or all of the primitives in a Joint. Connect to a Joint to see a list of its primitives.

Actuation

Enable	Primitive	Position	Units	Velocity
✓	R1	-60	deg	

Block Parameters: Joint2 Initial Condition

Joint Initial Condition

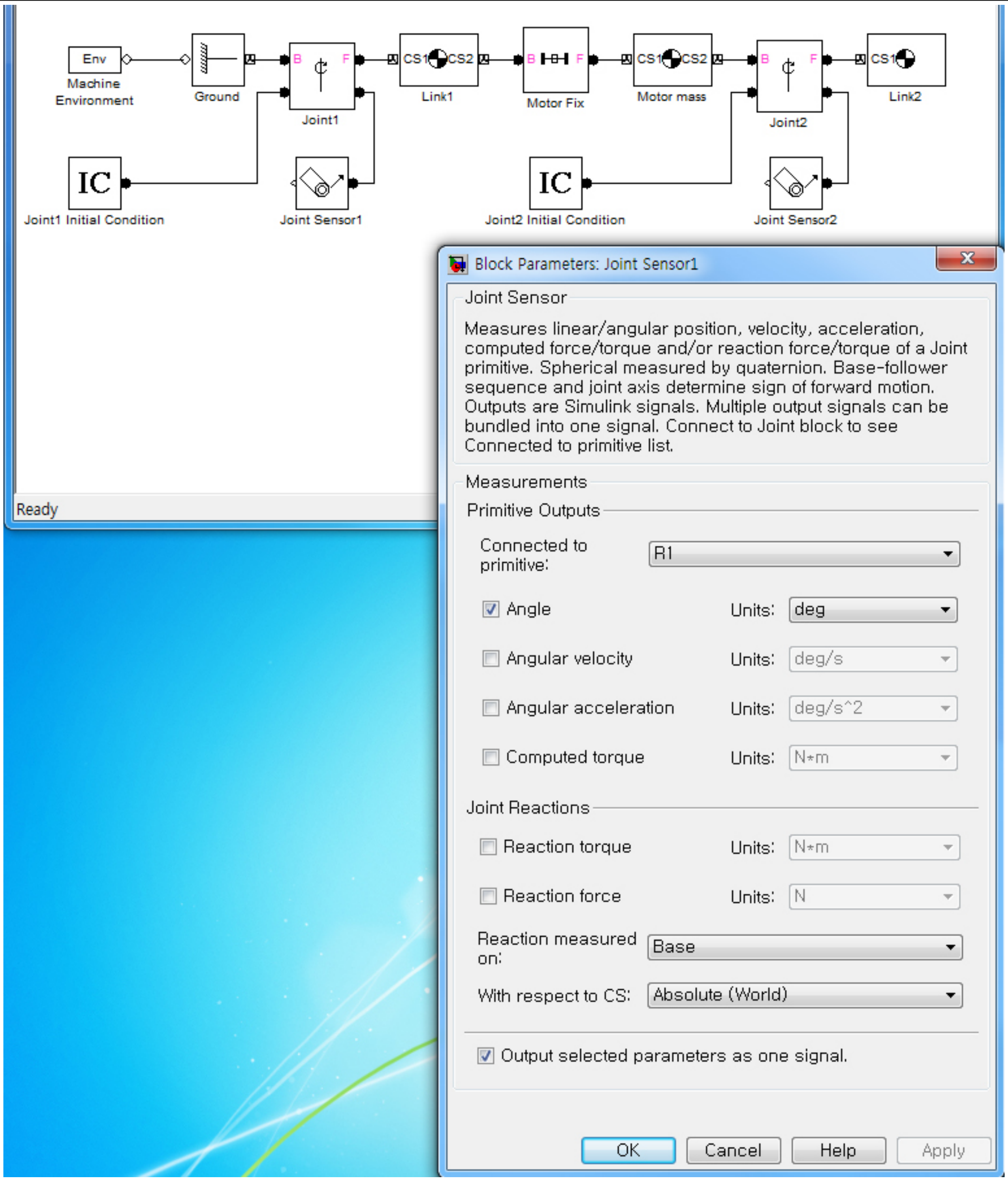
Sets the initial linear/angular position and velocity of some or all of the primitives in a Joint. Connect to a Joint to see a list of its primitives.

Actuation

Enable	Primitive	Position	Units	Velocity
☐	R1	0	deg	

Fig. 11: Joint Initial Condition block

- Next, add sensor to measure the joint displacement.
- Add Joint Sensor block in Sensors & Actuators library to the model and connect it to the corresponding Joint port. Check Angle and set Units to deg to measure joint displacement in degree.



Block Parameters: Joint Sensor1

Joint Sensor

Measures linear/angular position, velocity, acceleration, computed force/torque and/or reaction force/torque of a Joint primitive. Spherical measured by quaternion. Base-follower sequence and joint axis determine sign of forward motion. Outputs are Simulink signals. Multiple output signals can be bundled into one signal. Connect to Joint block to see Connected to primitive list.

Measurements

Primitive Outputs

Connected to primitive: R1

- [✓] Angle — Units: deg
- [ ] Angular velocity — Units: deg/s
- [ ] Angular acceleration — Units: deg/s^2
- [ ] Computed torque — Units: N*m

Joint Reactions

- [ ] Reaction torque — Units: N*m
- [ ] Reaction force — Units: N

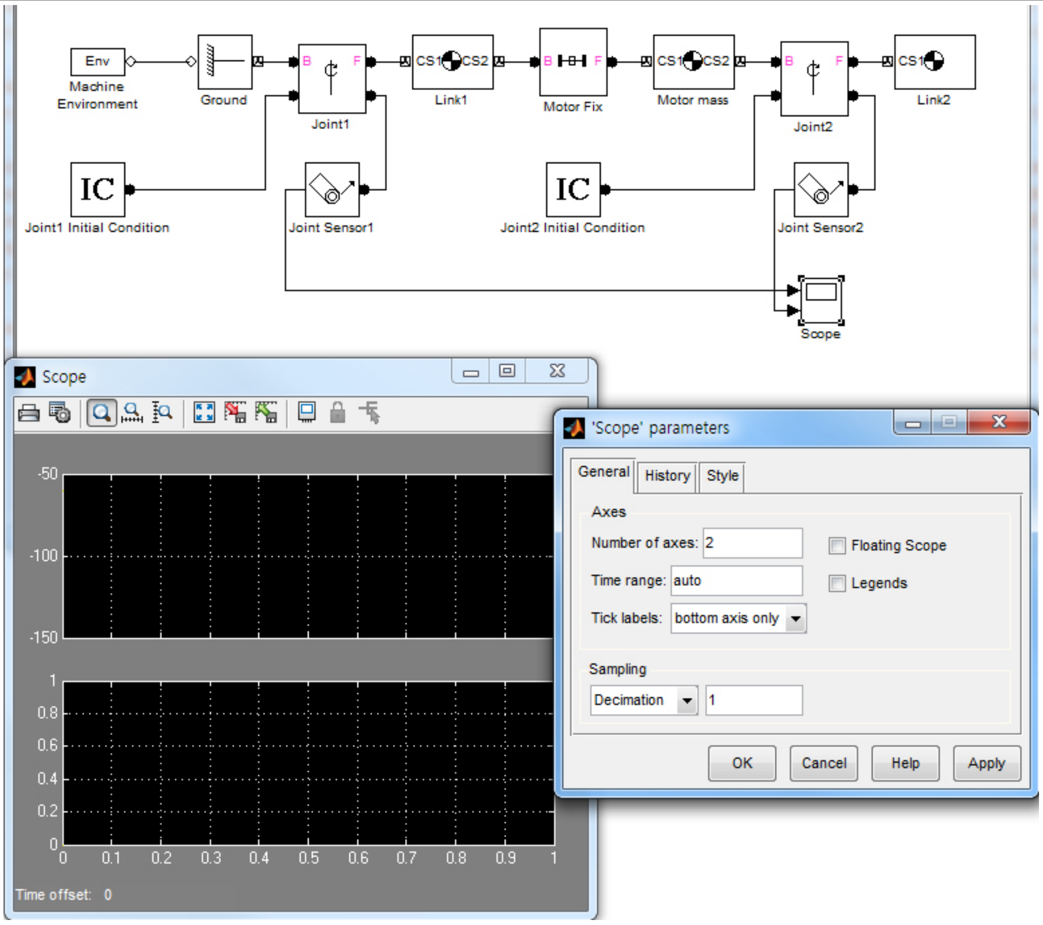
Reaction measured on: Base

With respect to CS: Absolute (World)

[✓] Output selected parameters as one signal.

Fig. 12: Joint Sensor block

- Add Scope block in Simulink → Sinks to the model to display the joint displacement.
- Click Scope block and Parameters. Set Number of axes to 2 to display two joint angles.



'Scope' parameters

Axes

Number of axes: 2

Time range: auto

Tick labels: bottom axis only

Sampling

Decimation 1

Fig. 13: Scope block

- Set Simulation Stop Time to 1 and run Forward Dynamic Simulation. The following simulation result can be obtained.

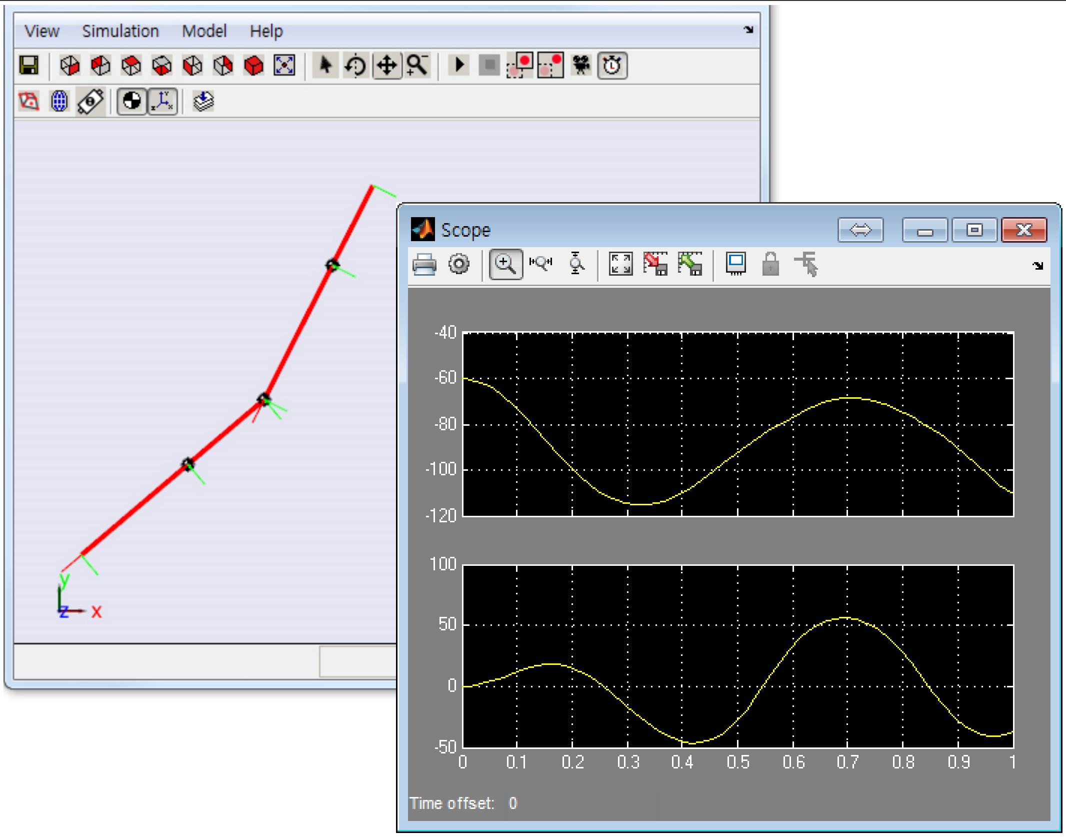


Fig. 14: Forward Dynamics Simulation Result

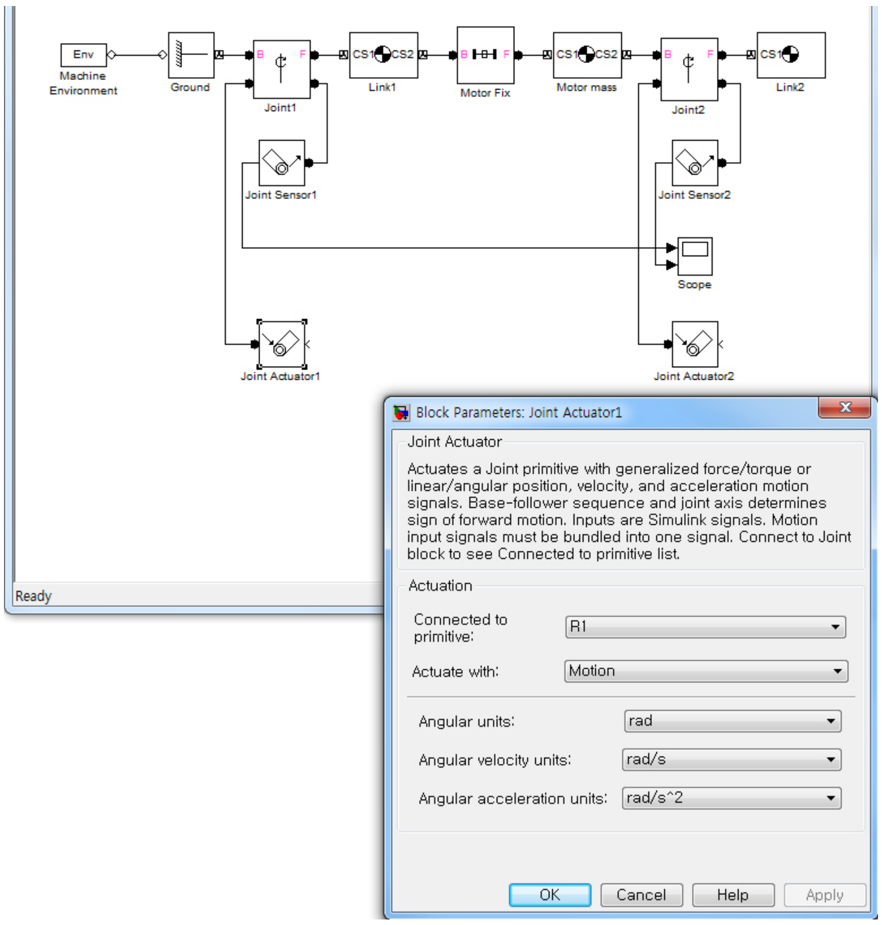
[2] Inverse Dynamics Simulation Practice of Planar Robot

(1) 2-DOF serial robot

- In this section, the inverse dynamics will be simulated in the joint space first and then the Cartesian space.

1) Inverse dynamics simulation in the joint space

- Let’s make the Inverse Dynamics Simulations model in the joint space from the previous model.
- In the previous model, delete Joint Initial Condition block, add Joint Actuator block in Sensors & Actuators library, and connect it to Joint block.
- Click Joint Actuator block, set Actuate with parameter to Motion and set the units to rad, rad/s and rad/s^2.



Joint Actuator

Actuates a Joint primitive with generalized force/torque or linear/angular position, velocity, and acceleration motion signals. Base-follower sequence and joint axis determines sign of forward motion. Inputs are Simulink signals. Motion input signals must be bundled into one signal. Connect to Joint block to see Connected to primitive list.

Actuation

Connected to primitive: R1

Actuate with: Motion

Angular units: rad

Angular velocity units: rad/s

Angular acceleration units: rad/s^2

Fig. 15: Joint Actuator block

- In Joint Actuator block, the position, velocity, and acceleration information is required.
- Let’s make the trajectory generation block with MATLAB Function block.
- Insert MATLAB Function block in Simulink → User-Defined Functions.
- Use the trajectory program in Chapter 1.
- Input parameters are dx (total displacement), tsec (Simulation Stop Time), and tout (Current Simulation Time).

<Trajectory Function>

```
function xd = fcn(dx, tsec, tout)

MAX_CH=2;           % Two Actuators %
D2R=pi/180;

% Initializing Variables %
xd=zeros(MAX_CH,1);
x0=D2R*[0,0]';
    [c] = coeff_3rd(x0, zeros(MAX_CH,1), x0+dx, zeros(MAX_CH,1), tsec);
    for ch=1:MAX_CH
        xd(ch,1)=c(ch,1)+c(ch,2)*tout+c(ch,3)*tout^2+c(ch,4)*tout^3;
    end

function [c] = coeff_3rd(sp, sv, ep, ev, tsec)
MAX_CH=2;
c=zeros(MAX_CH,4);
for ch=1:MAX_CH
    c(ch,1)=sp(ch,1);
    c(ch,2)=sv(ch,1);
    c(ch,3)=(3.0*(ep(ch,1)-sp(ch,1))-(2.0*sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec);
    c(ch,4)=(-2.0*(ep(ch,1)-sp(ch,1))+(sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec*tsec);
end
```

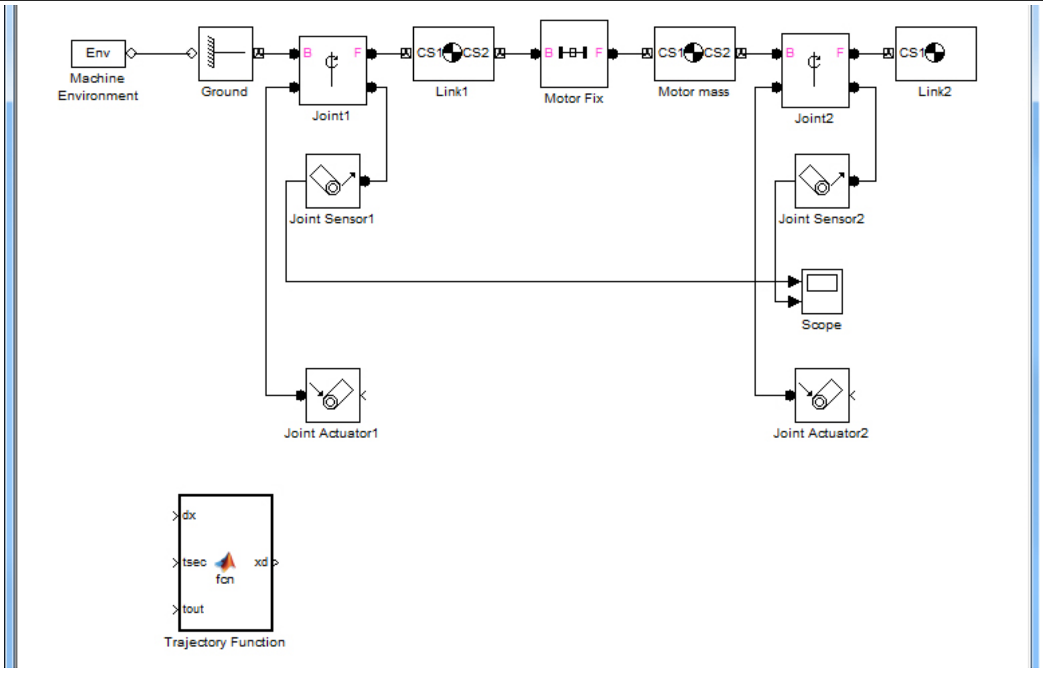
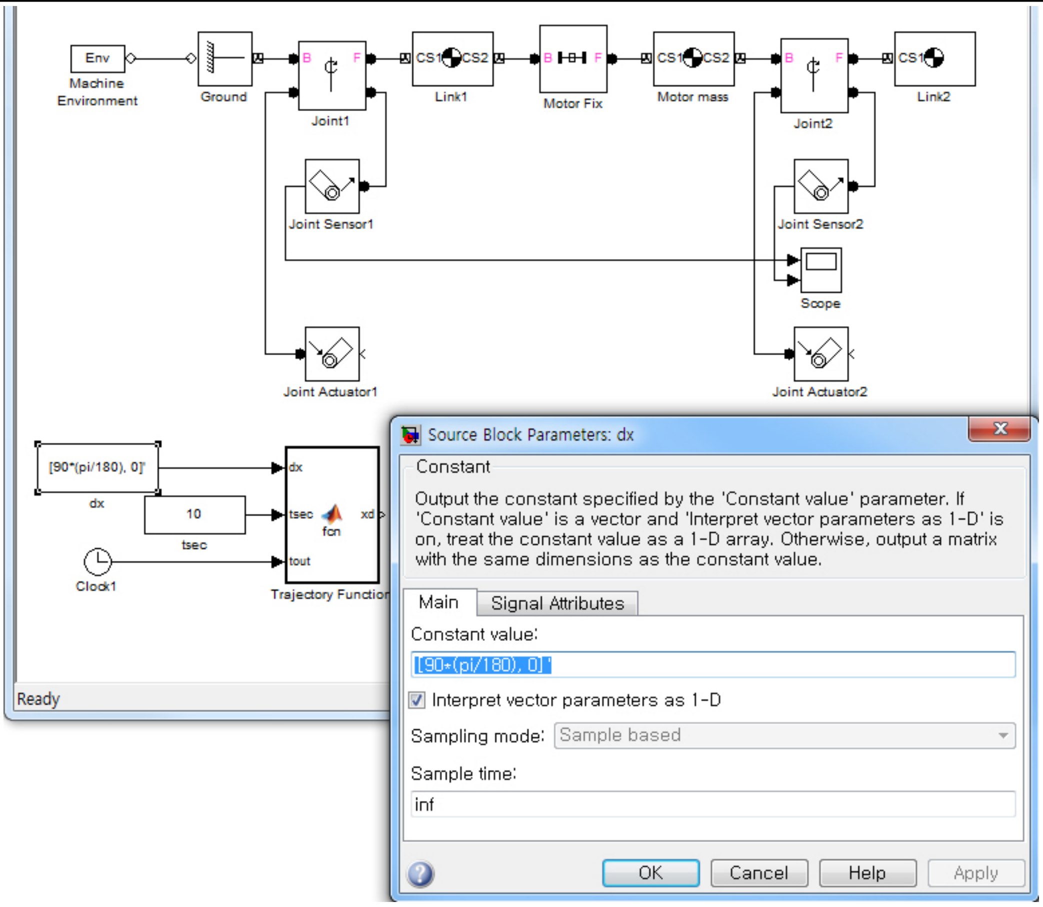


Fig. 16: Trajectory Function block

- Add Constant block and Clock block in Simulink → Sources to the model. Set ‘dx’ and ‘tsec’ values using Constant block, and connect Clock block to ‘tout’. Simulate Joint1 from 0 to 90° for 10 sec. Note the unit of ‘dx’ is the radian.



Source Block Parameters: dx

Constant

Output the constant specified by the 'Constant value' parameter. If 'Constant value' is a vector and 'Interpret vector parameters as 1-D' is on, treat the constant value as a 1-D array. Otherwise, output a matrix with the same dimensions as the constant value.

Main | Signal Attributes

Constant value: [90*(pi/180), 0]'

☑ Interpret vector parameters as 1-D

Sampling mode: Sample based

Sample time: inf

Fig. 17: Source block

- Use Demux block in Simulink → Signal Routing to separate 'xd' vector (Joint trajectory).
- Since the position, velocity, and acceleration data are required, take the derivatives of position by using Derivative block in Simulink → Continuous to get the velocity and acceleration data. Using Mux block in Simulink → Signal Routing, make the vector of the position, velocity, and acceleration.
- Create Subsystem by click and right mouse button as follow.

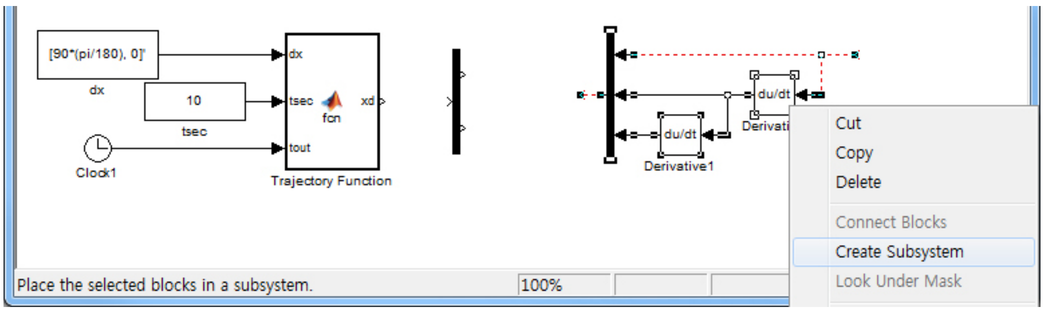


Fig. 18: Create Subsystem block

- Finally, change the outputs of Joint Sensor from Angle to Computed torque. Run the simulation for 10 sec. The following torque data from the Inverse Dynamics Simulation can be obtained.

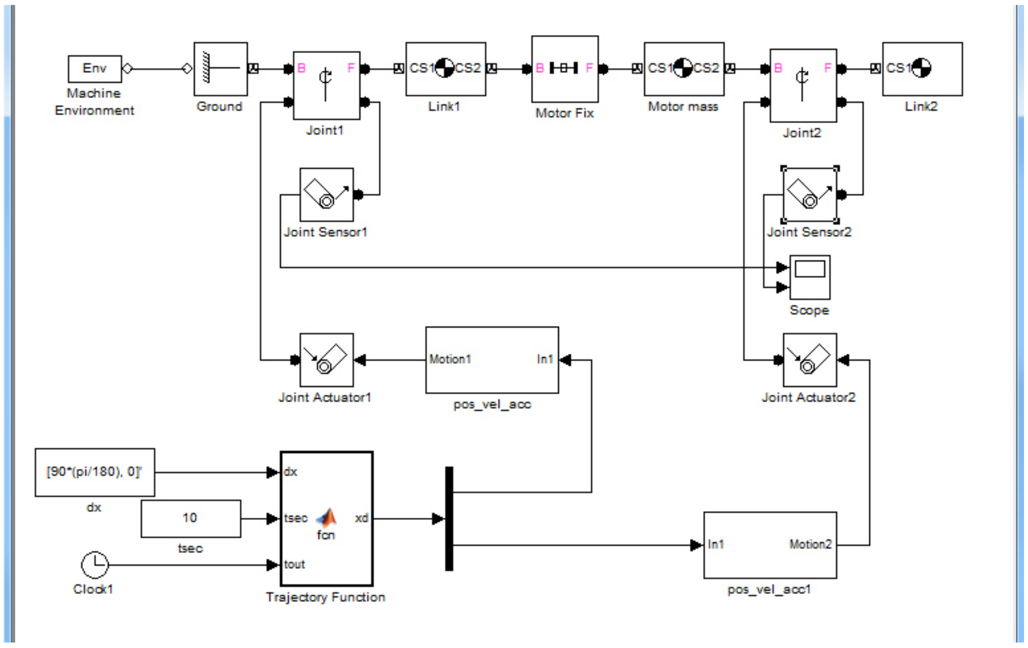


Fig. 19: Planar 2-DOF serial robot model

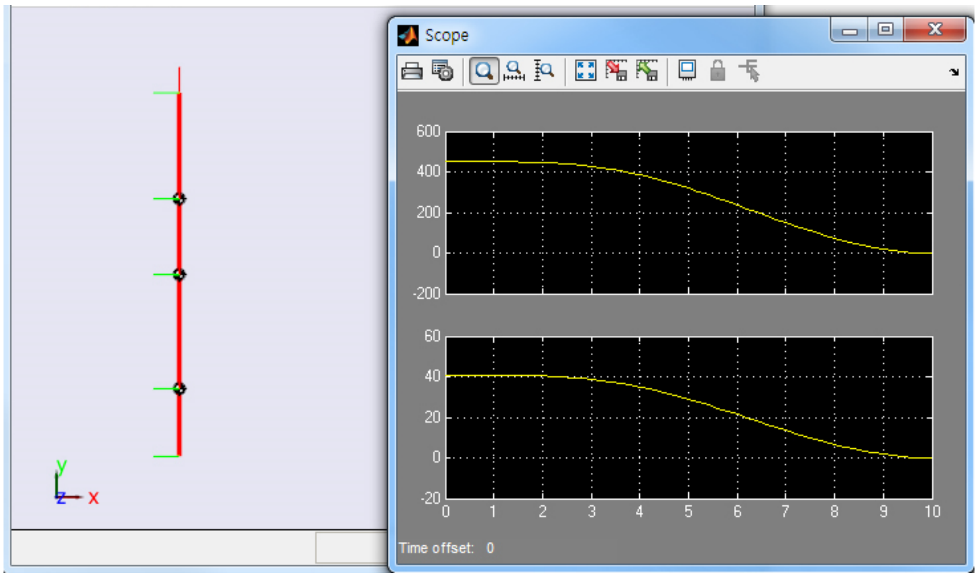


Fig. 20: Simulation result of the planar 2-DOF serial robot model

2) Inverse dynamics simulation in the Cartesian space

- From the previous model, let’s do the simulation in the Cartesian space by moving the end-effector.
- Insert MATLAB function block and copy the inverse kinematics function of the 2-DOF serial robot in Chapter 1.

<Inverse Kinematics Function>

```
function thd    = fcn(xd)

persistent index thd_p

% Initializing Variables %
if isempty(index), index=0; end
if isempty(thd_p), thd_p=(pi/180)*[0,90]'; end

w_index=1;   % In workspace %
a1=100; a2=100;

% (1) Calculate th2 %
px=xd(1,1); py=xd(2,1);
kapha=(px^2+py^2-a1^2-a2^2)/(2*a1*a2);
if abs(kapha)>1
       w_index=-1;   % Out of Workspace %
       kapha=1;
end
th2=+acos(kapha); % Elbow Down %
%th2=-acos(kapha); % Elbow Up %

% (2) Calculate th1 %
delta=a1^2+a2^2+2*a1*a2*cos(th2);
cth1=(px*(a1+a2*cos(th2))+py*a2*sin(th2))/delta;
sth1=(-px*a2*sin(th2)+py*(a1+a2*cos(th2)))/delta;
th1=atan2(sth1, cth1);
if th1<0, th1=th1+2*pi; end % 0<th<360 %

% (3) Check Joint Limits %
if th1<0 || th1>pi, w_index=-1; end
if th2<pi/6 || th2>5*pi/6, w_index=-1; end

% Result %
if w_index==1
       thd=[th1, th2]';
else
       thd=thd_p;
end

% Save the current value %
thd_p=thd;
```

- Note that ‘dx’ is the end-effector position. The trajectory function needs to be modified as follow.
- Note that the initial end-effector position is set to [100, 100].

<Trajectory Function>

```
function xd = fcn(dx, tsec, tout)

MAX_CH=2;           % Two Actuators %
a1=100; a2=100;

% Initializing Variables %
%if isempty(xd_p), xd_p=zeros(MAX_CH,1); end
xd=zeros(MAX_CH,1);
x0=[a1,a2]';

%tout=index*dt;

        [c] = coeff_3rd(x0, zeros(MAX_CH,1), x0+dx, zeros(MAX_CH,1), tsec);
        for ch=1:MAX_CH
                xd(ch,1)=c(ch,1)+c(ch,2)*tout+c(ch,3)*tout^2+c(ch,4)*tout^3;
        end

function [c] = coeff_3rd(sp, sv, ep, ev, tsec)
MAX_CH=2;
c=zeros(MAX_CH,4);
for ch=1:MAX_CH
        c(ch,1)=sp(ch,1);
        c(ch,2)=sv(ch,1);
        c(ch,3)=(3.0*(ep(ch,1)-sp(ch,1))-(2.0*sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec);
        c(ch,4)=(-2.0*(ep(ch,1)-sp(ch,1))+(sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec*tsec);
end
```

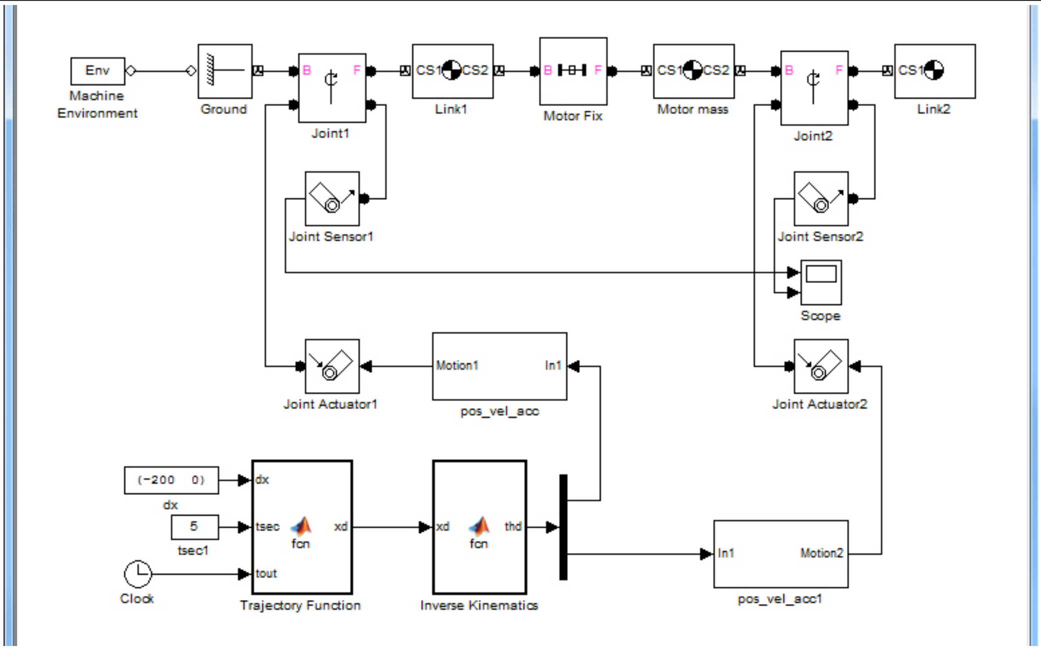


Fig. 21: Planar 2-DOF serial robot model

- From the initial position, the end-effector is moved by 200mm along the –X direction for 5 sec. The actuator torques can be obtained as follow.

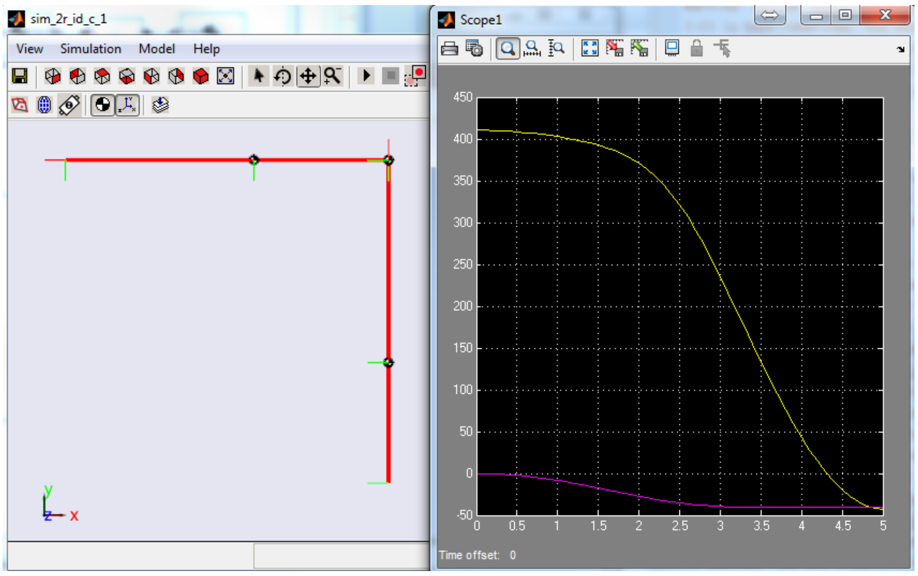
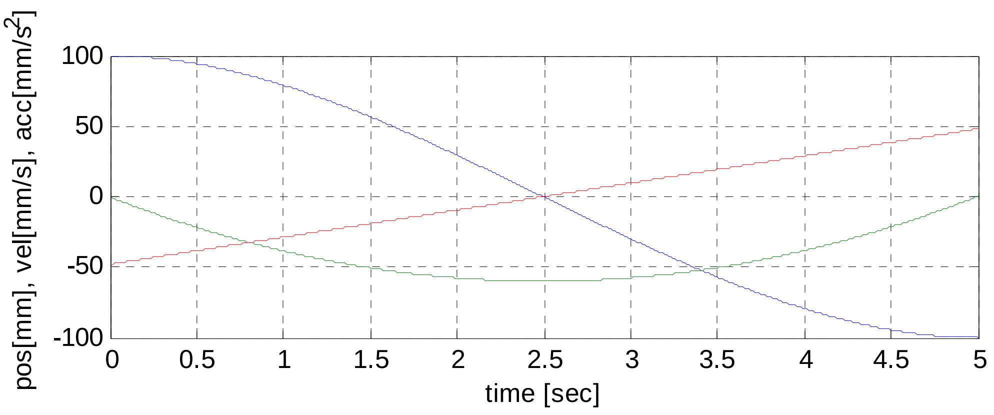
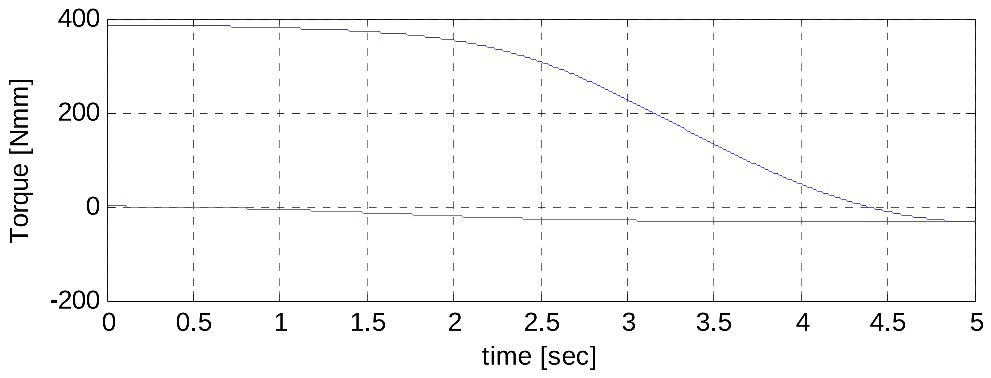


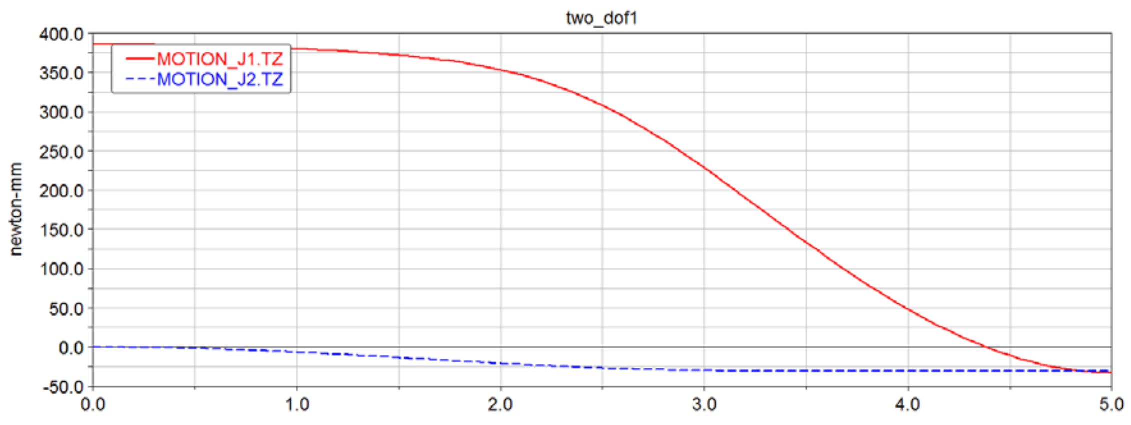
Fig. 22: Simulation result of planar 2-DOF serial robot model

[Comparison with the dynamic equation result]



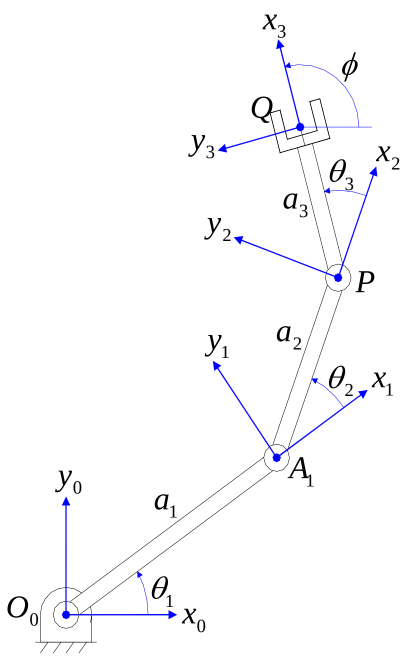


[Comparison with ADAMS simulation result]



- It is noted that the SimMechanics, dynamic equation, and ADAMS results are the same.

(2) 3-DOF serial robot



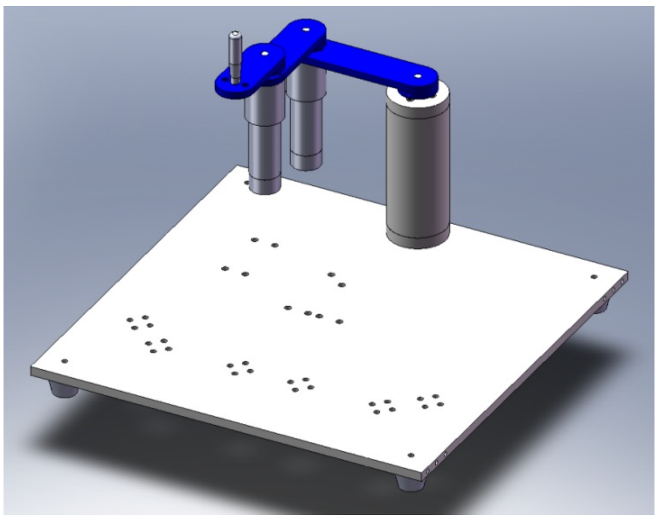


Fig. 23: Planar 3-DOF serial robot.

1) Parameters

	Mass(kg)	Inertia(kg mm^2)	CG Position(m) /from CS1	CS(m) /from World
Ground	-	-	-	[0 0 0]
Link1	0.071	89.395	[0.0373 0 0]	CS1 = [0 0 0] CS2 = [0.1 0 0]
Motor2	0.292	0	[0 0 0]	CS1 = [0.1 0 0] CS2 = [0.1 0 0]
Link2	0.055	38.709	[0.0236 0 0]	CS1 = [0.1 0 0] CS2 = [0.17 0 0]
Motor3	0.292	0	[0 0 0]	CS1 = [0.17 0 0] CS2 = [0.17 0 0]
Link3	0.05	24.587	[0.0186 0 0]	CS1 = [0.17 0 0] CS2 = [0.22 0 0]

- Link lengths: $a_1 = 100$ mm, $a_2 = 70$ mm, $a_3 = 50$ mm

2) Inverse dynamics simulation

- Add one more actuator and link to the previous 2-DOF serial robot model. And change the trajectory function and inverse kinematics function as follows.

<Trajectory Function>

```
function xd = fcn(dx, tsec, tout)

MAX_CH=3;              % Two Actuators %
a1=100; a2=70; a3=50;

% Initializing Variables %
xd=zeros(MAX_CH,1);
x0=[a1,a2+a3, pi/2]';

%tout=index*dt;

    [c] = coeff_3rd(x0, zeros(MAX_CH,1), x0+dx, zeros(MAX_CH,1), tsec);
    for ch=1:MAX_CH
        xd(ch,1)=c(ch,1)+c(ch,2)*tout+c(ch,3)*tout^2+c(ch,4)*tout^3;
    end

function [c] = coeff_3rd(sp, sv, ep, ev, tsec)
MAX_CH=3;
c=zeros(MAX_CH,4);
for ch=1:MAX_CH
    c(ch,1)=sp(ch,1);
    c(ch,2)=sv(ch,1);
    c(ch,3)=(3.0*(ep(ch,1)-sp(ch,1))-(2.0*sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec);
    c(ch,4)=(-2.0*(ep(ch,1)-sp(ch,1))+(sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec*tsec);
end
```

<Inverse Kinematics Function>

```
function thd    = fcn(xd)

persistent index thd_p

% Initializing Variables %
if isempty(index), index=0; end
if isempty(thd_p), thd_p=(pi/180)*[0,90,0]'; end

w_index=1;    % In workspace %
a1=100; a2=70; a3=50;

qx=xd(1,1); qy=xd(2,1); phi=xd(3,1);

% (1) Calculate th2 %
px=qx-a3*cos(phi); py=qy-a3*sin(phi);
kapha=(px^2+py^2-a1^2-a2^2)/(2*a1*a2);
if abs(kapha)>1
    w_index=-1;    % Out of Workspace %
    kapha=1;
end
th2=+acos(kapha); % Elbow Down %
```

```
%th2=-acos(kapha); % Elbow Up %

% (2) Calculate th1 %
delta=a1^2+a2^2+2*a1*a2*cos(th2);
cth1=(px*(a1+a2*cos(th2))+py*a2*sin(th2))/delta;
sth1=(-px*a2*sin(th2)+py*(a1+a2*cos(th2)))/delta;
th1=atan2(sth1, cth1);
if th1<0, th1=th1+2*pi; end % 0<th<360 %

% (3) Calcuate th3 %
th3=phi-(th1+th2);

% (3) Check Joint Limits %
if th1<0 || th1>pi, w_index=-1; end
if th2<pi/6 || th2>5*pi/6, w_index=-1; end

% Result %
if w_index==1
      thd=[th1, th2, th3]';
else
      thd=thd_p;
end

% Save the current value %
thd_p=thd;
```

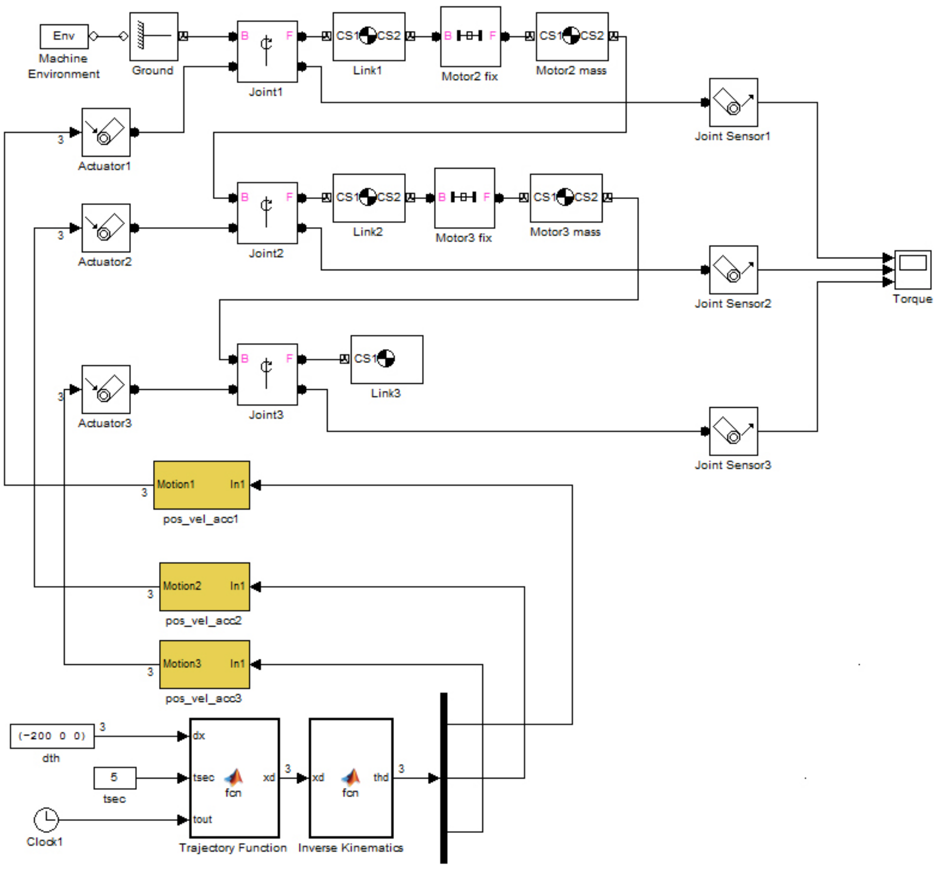


Fig. 24: Planar 3-DOF serial robot model.

- When the end-effector is moved from [100, 120] to [-100, 120] for 5 sec, the actuator torques can be obtained from the inverse dynamics simulation.

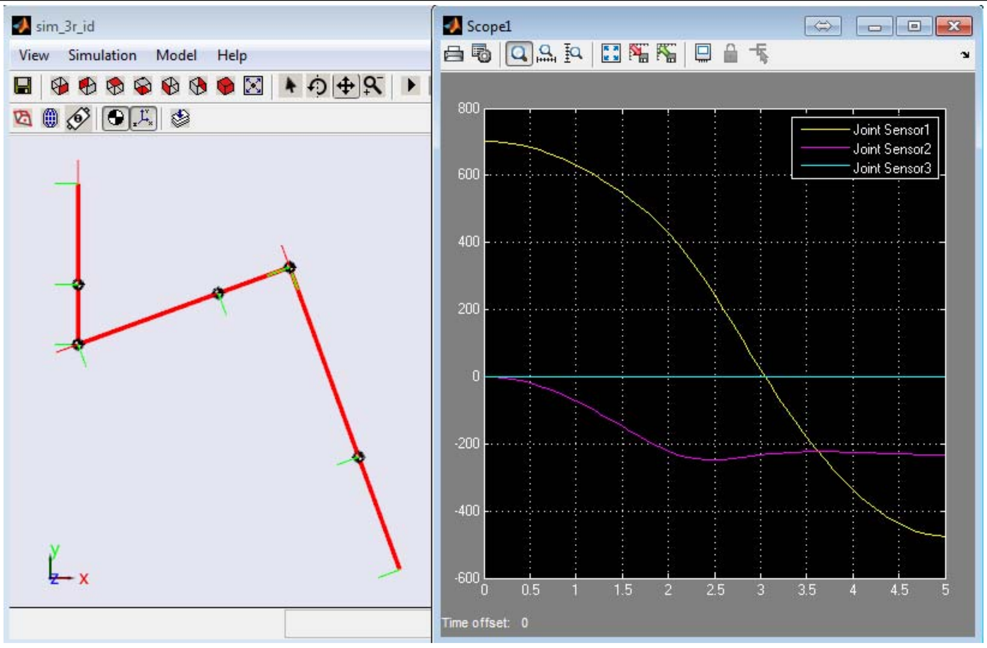
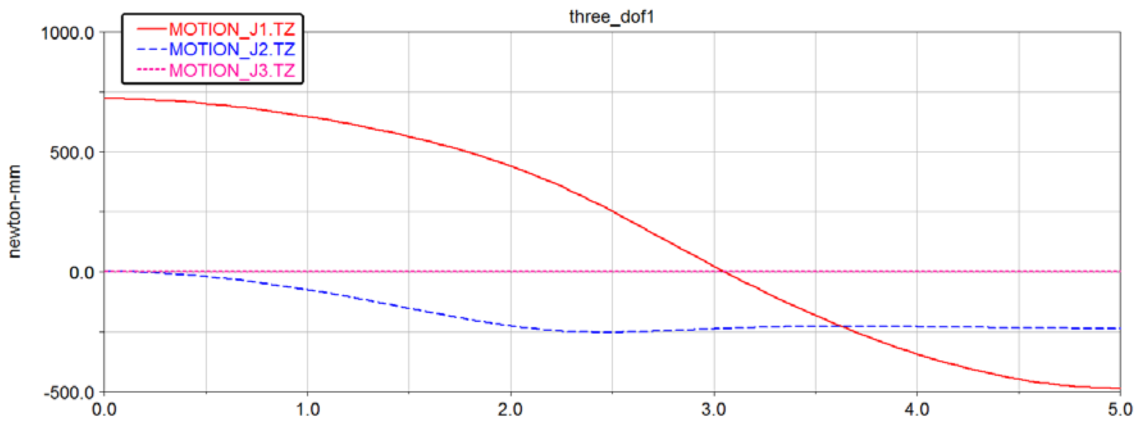


Fig. 25: Simulation result of the planar 3-DOF serial robot model.

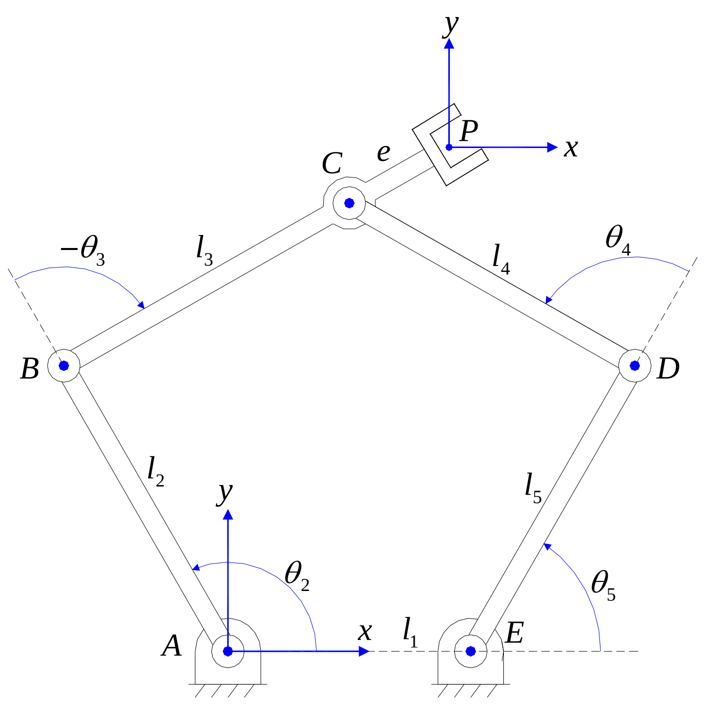
[Comparison with ADAMS simulation result]



(3) 5-bar (Type I) robot

1) Parameters

	Mass(kg)	Inertia (kg mm$^{2)}$	CG Position(m) /from CS1	CS(m) /from World
L_Ground	-	-	-	[0 0 0]
L_Link1	0.053	110.61	[-0.025592 0.046537 0]	CS1 = [0 0 0] CS2 = [-0.048187 0.087624 0]
L_Link2	0.054	120.188	[0.042537 0.026153 0]	CS1 = [-0.048187 0.087624 0] CS2 = [0.037 0.140 0] CS3 = [0.060852 0.154665 0]
R_Ground	-	-	-	[0.074 0 0]
R_Link1	0.061	134.303	[0.022017 0.040035 0]	CS1 = [0.074 0 0] CS2 = [0.122187 0.087624 0]
R_Link2	0.045	90.992	[-0.042594 0.026188 0]	CS1 = [0.122187 0.087624 0] CS2 = [0.037 0.140 0]



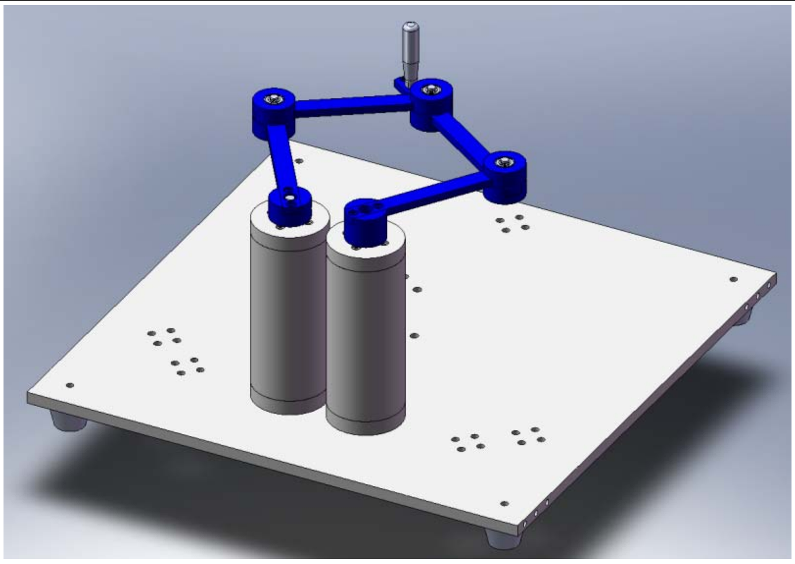
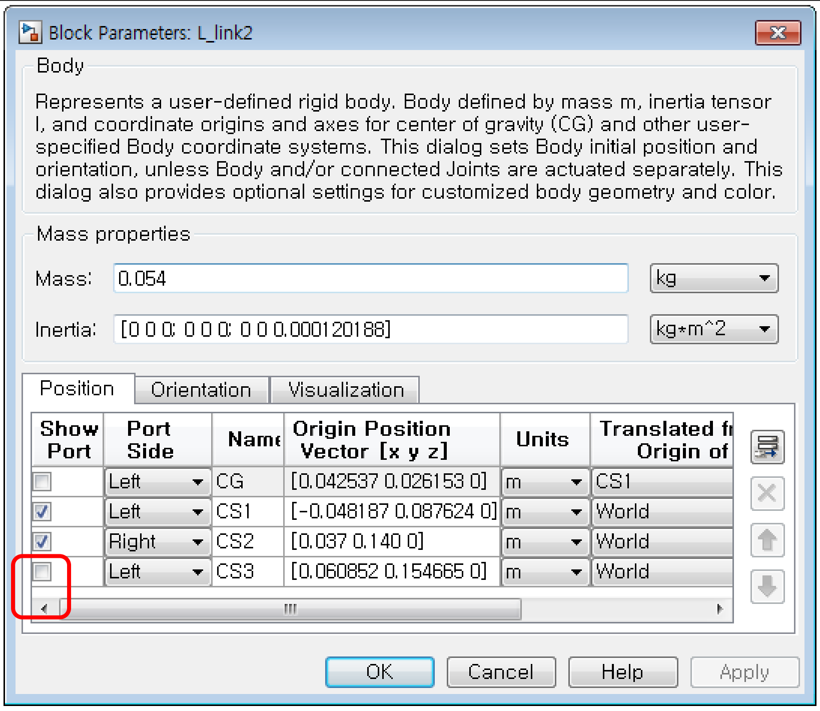


Fig. 26: Planar 5-bar (Type I) robot.

- Kinematic parameters: $l_1 = \overline{AE} = 74$, $l_2 = \overline{AB} = 100$, $l_3 = \overline{BC} = 100 + 28 = 128$, $l_4 = \overline{CD} = 100$, $l_5 = \overline{DE} = 100$, $e = \overline{CP} = 28$ mm

2) Inverse dynamics simulation

- In a similar manner to 2-DOF and 3-DOF serial robots, the 5-bar robot can be modelled except some points.
- Since the 5-bar robot is a closed-loop mechanism, two Ground blocks are required. However, the Machine Environment block is connected to only one Ground block. So, uncheck Show Machine Environment port in the second Ground block.
- To model the end-effector point P, add CS3 line in L_Link2 block.


Block Parameters: L_link2

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass: 0.054 kg

Inertia: [0 0 0; 0 0 0; 0 0 0.000120188] kg*m^2

Position | Orientation | Visualization

Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated fr Origin of
☐	Left	CG	[0.042537 0.026153 0]	m	CS1
☑	Left	CS1	[-0.048187 0.087624 0]	m	World
☑	Right	CS2	[0.037 0.140 0]	m	World
☐	Left	CS3	[0.060852 0.154665 0]	m	World

Fig. 27: Add Port Line

<Trajectory Function>

```
function xd = fcn(dx, tsec, tout)

MAX_CH=2;            % Two Actuators %

% Initializing Variables %
xd=zeros(MAX_CH,1);
x0=[60.852, 154.665]';

[c] = coeff_3rd(x0, zeros(MAX_CH,1), x0+dx, zeros(MAX_CH,1), tsec);
for ch=1:MAX_CH
        xd(ch,1)=c(ch,1)+c(ch,2)*tout+c(ch,3)*tout^2+c(ch,4)*tout^3;
end

function [c] = coeff_3rd(sp, sv, ep, ev, tsec)
MAX_CH=2;
c=zeros(MAX_CH,4);
for ch=1:MAX_CH
        c(ch,1)=sp(ch,1);
        c(ch,2)=sv(ch,1);
        c(ch,3)=(3.0*(ep(ch,1)-sp(ch,1))-(2.0*sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec);
        c(ch,4)=(-2.0*(ep(ch,1)-sp(ch,1))+(sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec*tsec);
end
```

<Inverse Kinematics Function>

```
function thd    = fcn(xd)

persistent index thd_p
persistent th3_p th4_p

% Initializing Variables %
if isempty(index), index=0; end
if isempty(thd_p), thd_p=[2.07359,1.06801]'; end
if isempty(th3_p), th3_p=0; end
if isempty(th4_p), th4_p=0; end

% Kinematic Parameters %
L1=74; L2=100; L3=100; L4=100; L5=100; L6=28;

w_index=1;    % In workspace %

% (1) Left Side 2-DOF Arm %
px=xd(1,1); py=xd(2,1);
k1=(px^2+py^2-L2^2-(L3+L6)^2)/(2*L2*(L3+L6));
if abs(k1)<=1
        th3=-acos(k1);
else
        w_index=-1;
        th3=th3_p;
end

DEL1=L2^2+(L3+L6)^2+2*L2*(L3+L6)*cos(th3);
cth2=(px*(L2+(L3+L6)*cos(th3))+py*(L3+L6)*sin(th3))/DEL1;
sth2=(-px*(L3+L6)*sin(th3)+py*(L2+(L3+L6)*cos(th3)))/DEL1;
th2=atan2(sth2,cth2);

% (2) Right Side 2-DOF Arm %
cx=px-L6*cos(th2+th3)-L1; cy=py-L6*sin(th2+th3);
k2=(cx^2+cy^2-L5^2-L4^2)/(2*L5*L4);
if abs(k2)<=1
        th4=acos(k2);
else
        w_index=-2;
        th4=th4_p;
end

DEL2=L5^2+L4^2+2*L5*L4*cos(th4);
cth5=(cx*(L5+L4*cos(th4))+cy*L4*sin(th4))/DEL2;
sth5=(-cx*L4*sin(th4)+cy*(L5+L4*cos(th4)))/DEL2;
th5=atan2(sth5,cth5);

% Result %
if w_index==1
        thd=[th2, th5]';
else
        thd=thd_p;
```

```
end

% Save the current value %
thd_p=thd;
th3_p=th3; th4_p=th4;
```

- The actuated joint offsets ([2.07359, 1.06801] [rad]) should be considered.
- Add Sum block in Simulink → Math Operations to make offset.

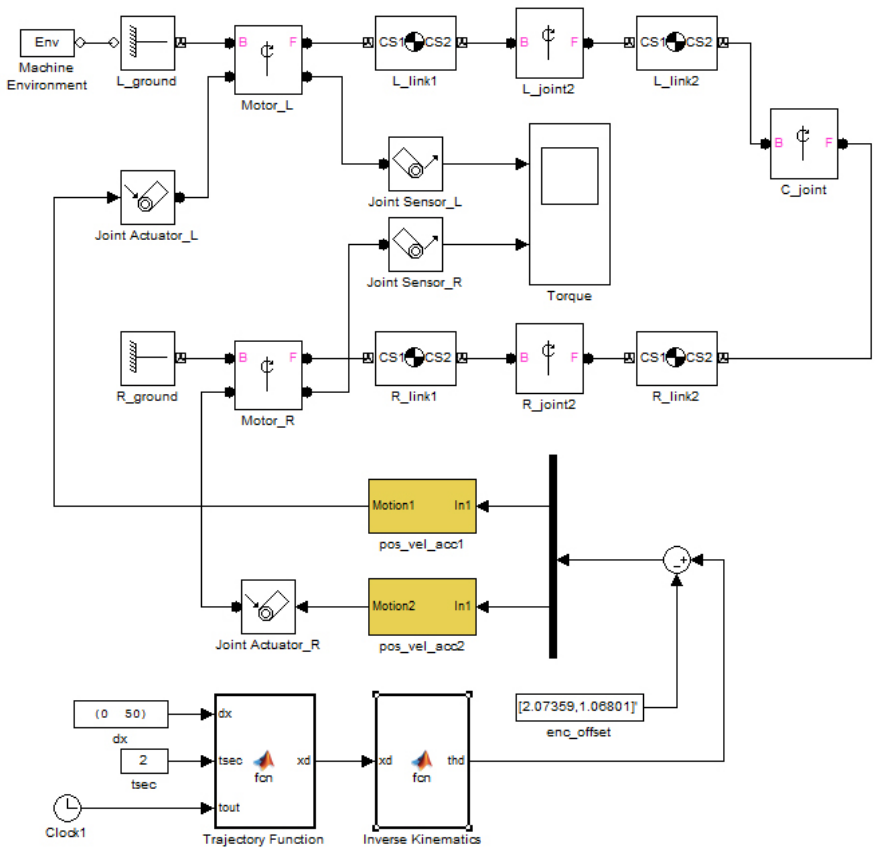


Fig. 28: 5-bar (Type I) robot model.

- From the initial end-effector position ([60.852, 154.665] [mm]), the end-effector is moved by 50mm along the X and Y axes for 2 sec, respectively. The joint torques can be obtained from the inverse dynamics simulation.

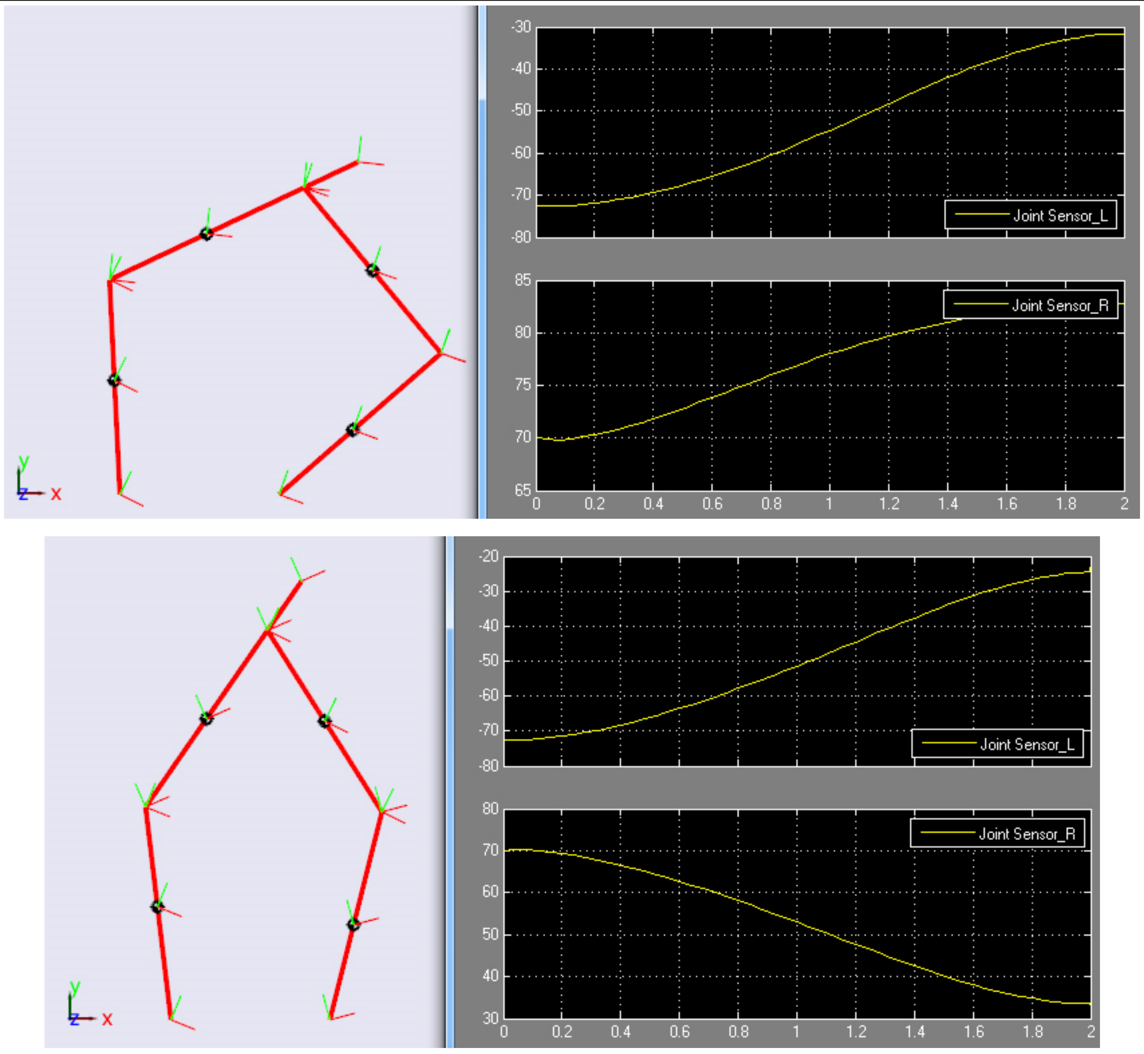
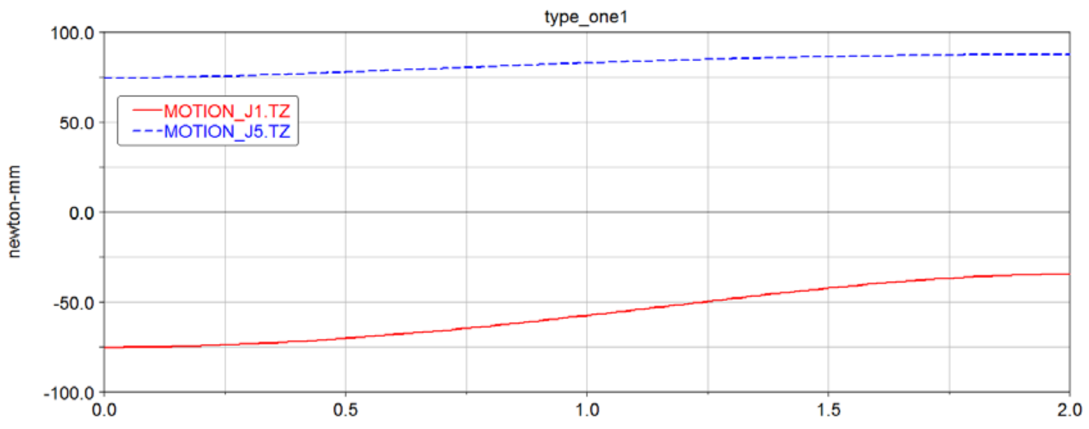
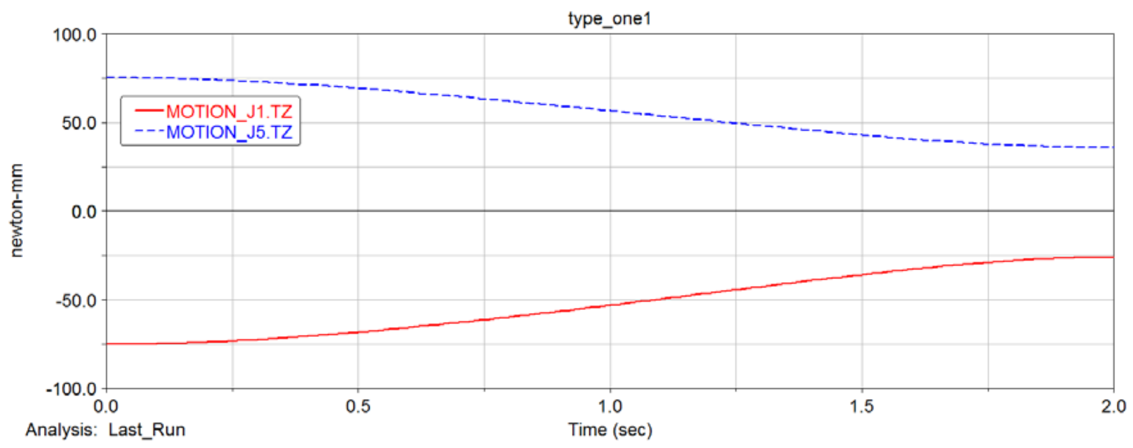


Fig. 2.29: Simulation Results of 5-bar Mechanism Type1 model.

[ADAMS simulation result (move along the X axis)]

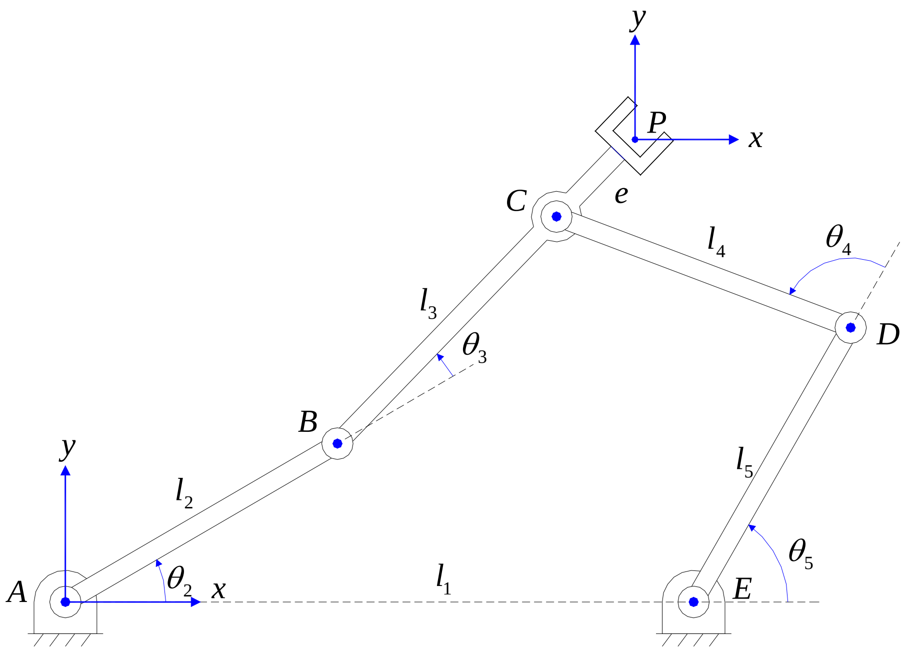


[ADAMS simulation result (move along the Y axis)]



(4) 5-bar (Type II) robot

1) Parameters



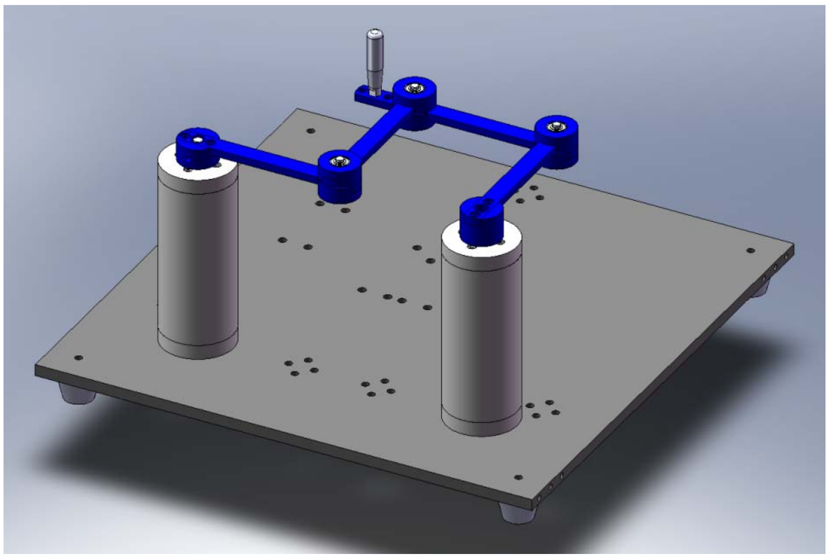


Fig. 30: Planar 5-bar (Type II) robot.

	Mass(kg)	Inertia(kg mm^{2})	CG Position(m) /from CS1	CS(m) /from World
L_Ground	-	-	-	[0 0 0]

L_Link1	0.053	110.621	[0.05311 0 0]	CS1 = [0 0 0]
				CS2 = [0.1 0 0]

L_Link2	0.054	120.188	[0 0.05992 0]	CS1 = [0.1 0 0]
				CS2 = [0.1 0.1 0]
				CS3 = [0.1 0.128 0]

R_Ground	-	-	-	[0.2 0 0]

R_Link1	0.061	134.303	[0 0.04569 0]	CS1 = [0.2 0 0]
				CS2 = [0.2 0.1 0]

R_Link2	0.045	90.992	[-0.05 0 0]	CS1 = [0.2 0.1 0]
				CS2 = [0.1 0.1 0]

- Kinematic parameters: $l_1 = \overline{AE} = 200$, $l_2 = \overline{AB} = 100$, $l_3 = \overline{BC} = 100 + 28 = 128$, $l_4 = \overline{CD} = 100$, $l_5 = \overline{DE} = 100$, $e = \overline{CP} = 28$ mm

2) Inverse dynamics simulation

- In a similar manner to Type I, the 5-bar (Type II) robot model can be made.

<Trajectory Function>

```
function xd = fcn(dx, tsec, tout)

MAX_CH=2;           % Two Actuators %

% Initializing Variables %
xd=zeros(MAX_CH,1);
x0=[100, 128]';

[c] = coeff_3rd(x0, zeros(MAX_CH,1), x0+dx, zeros(MAX_CH,1), tsec);
for ch=1:MAX_CH
    xd(ch,1)=c(ch,1)+c(ch,2)*tout+c(ch,3)*tout^2+c(ch,4)*tout^3;
end

function [c] = coeff_3rd(sp, sv, ep, ev, tsec)
MAX_CH=2;
c=zeros(MAX_CH,4);
for ch=1:MAX_CH
    c(ch,1)=sp(ch,1);
    c(ch,2)=sv(ch,1);
    c(ch,3)=(3.0*(ep(ch,1)-sp(ch,1))-(2.0*sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec);
    c(ch,4)=(-2.0*(ep(ch,1)-sp(ch,1))+(sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec*tsec);
end
```

<Inverse Kinematics Function>

```
function thd   = fcn(xd)

persistent index thd_p
persistent th3_p th4_p

% Initializing Variables %
if isempty(index), index=0; end
if isempty(thd_p), thd_p=(pi/180)*[0,90]'; end
if isempty(th3_p), th3_p=0; end
if isempty(th4_p), th4_p=0; end

% Kinematic Parameters %
L1=200; L2=100; L3=100; L4=100; L5=100; L6=28;

w_index=1;   % In workspace %

% (1) Left Side 2-DOF Arm %
px=xd(1,1); py=xd(2,1);
k1=(px^2+py^2-L2^2-(L3+L6)^2)/(2*L2*(L3+L6));
if abs(k1)<=1
      th3=+acos(k1);   % Note the sign %
else
      w_index=-1;
      th3=th3_p;
end

DEL1=L2^2+(L3+L6)^2+2*L2*(L3+L6)*cos(th3);
cth2=(px*(L2+(L3+L6)*cos(th3))+py*(L3+L6)*sin(th3))/DEL1;
sth2=(-px*(L3+L6)*sin(th3)+py*(L2+(L3+L6)*cos(th3)))/DEL1;
th2=atan2(sth2,cth2);

% (2) Right Side 2-DOF Arm %
cx=px-L6*cos(th2+th3)-L1; cy=py-L6*sin(th2+th3);
k2=(cx^2+cy^2-L5^2-L4^2)/(2*L5*L4);
if abs(k2)<=1
      th4=acos(k2);
else
      w_index=-2;
      th4=th4_p;
end

DEL2=L5^2+L4^2+2*L5*L4*cos(th4);
cth5=(cx*(L5+L4*cos(th4))+cy*L4*sin(th4))/DEL2;
sth5=(-cx*L4*sin(th4)+cy*(L5+L4*cos(th4)))/DEL2;
th5=atan2(sth5,cth5);

% Result %
if w_index==1
      thd=[th2, th5]';
else
      thd=thd_p;
```

```
end

% Save the current value %
thd_p=thd;
th3_p=th3; th4_p=th4;
```

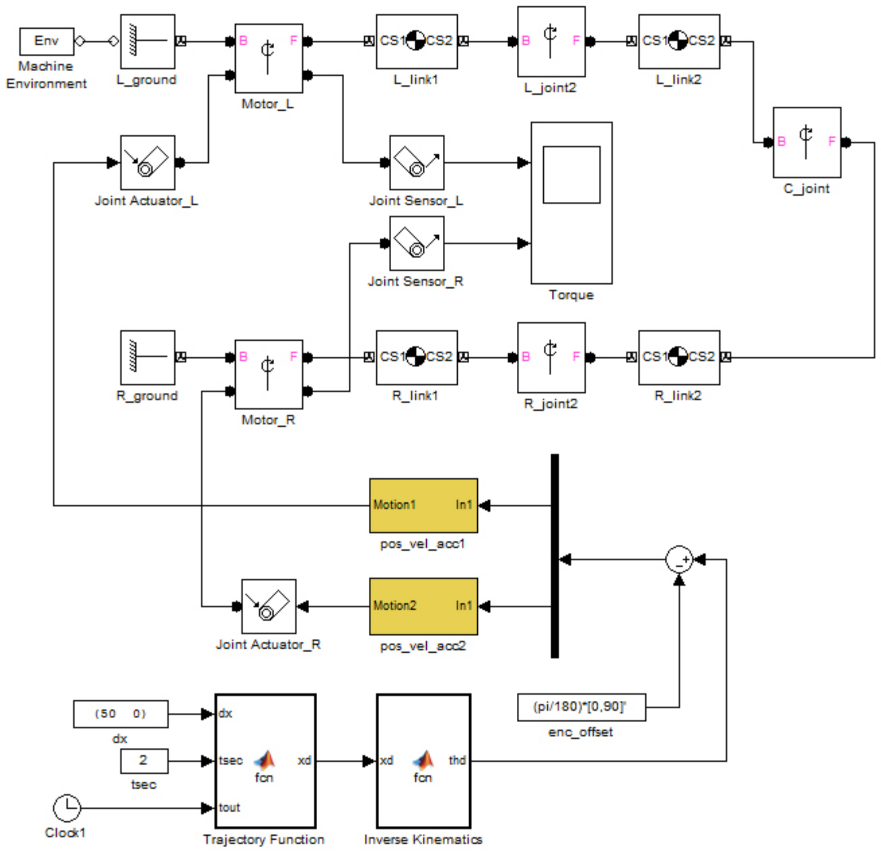


Fig. 31: 5-bar (Type II) robot model.

- Form the initial end-effector point ([100, 128]mm), the end-effector is moved by 50mm along the X and Y axes for 2 sec. The inverse dynamics results are as follows.

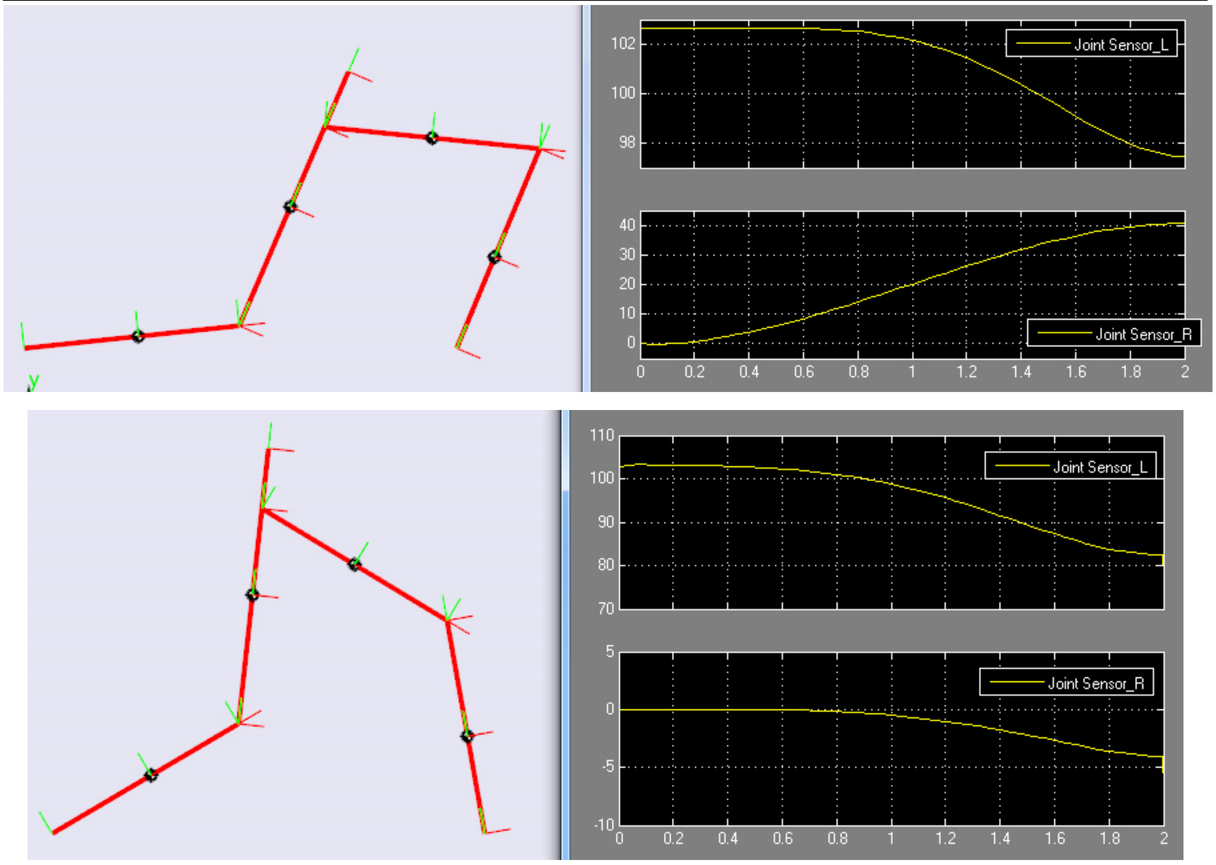
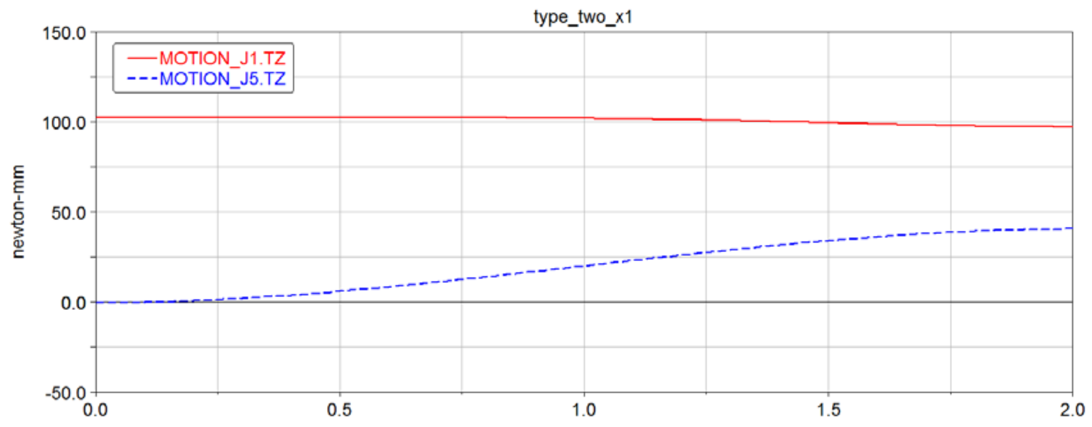
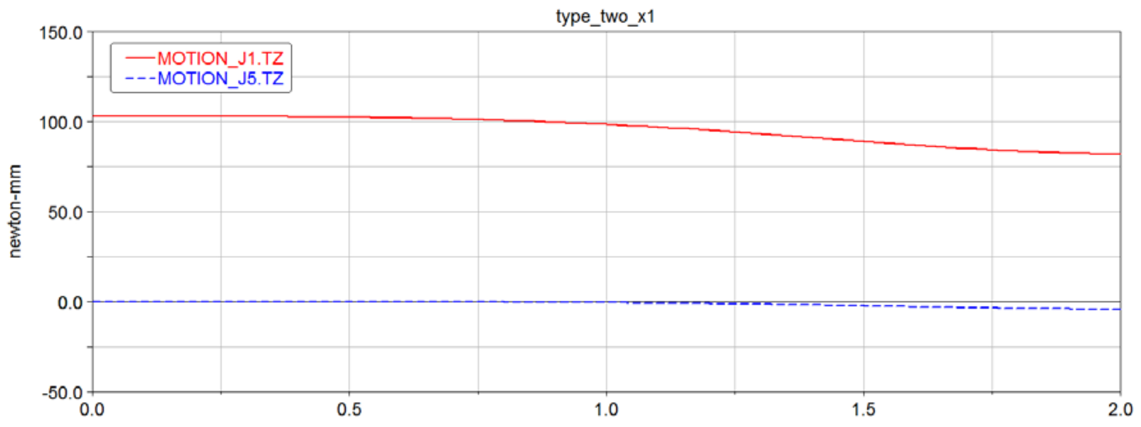


Fig. 32: Simulation results of 5-bar (Type II) robot model.

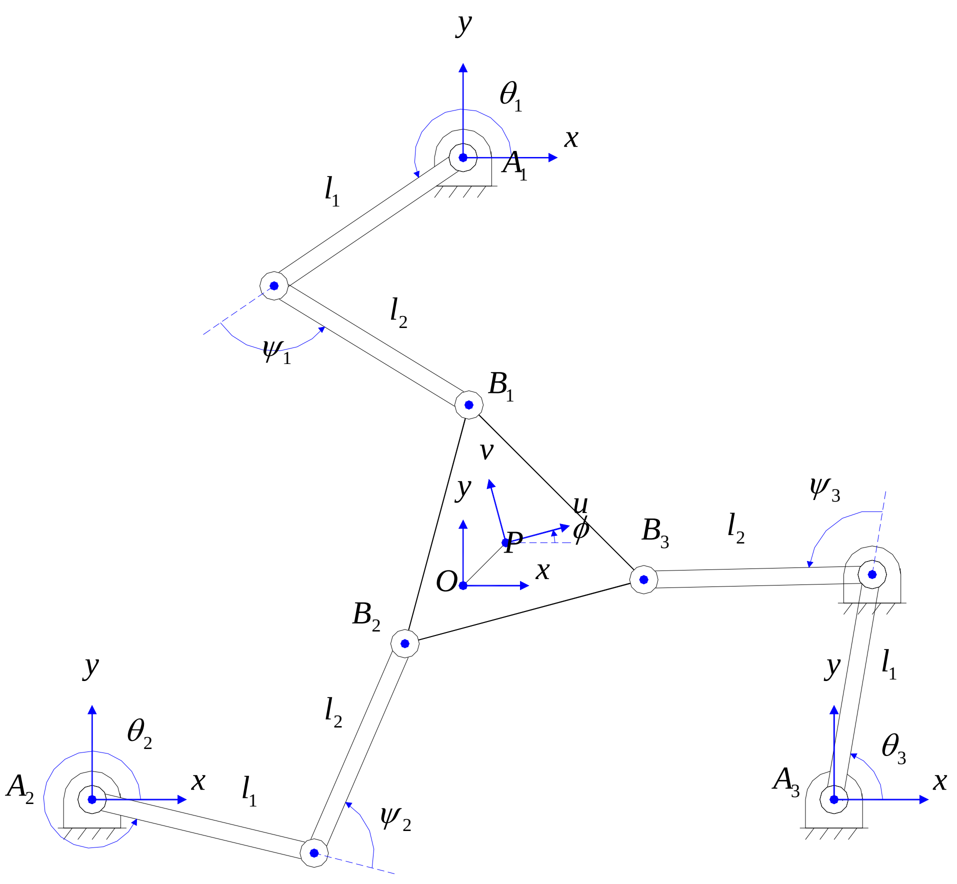
[ADAMS simulation result (move along the X axis)]



[ADAMS simulation result (move along the Y axis)]



(5) 3-DOF parallel robot



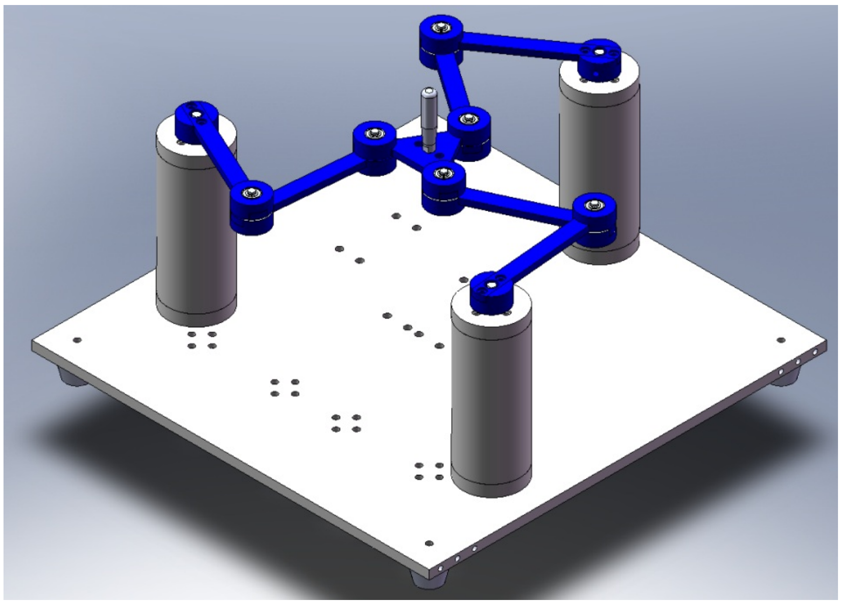


Fig. 33: Planar 3-DOF 3-RRR parallel robot.

- Kinematic parameters: $a = \overline{OA_i} = 150$ mm, $b = \overline{PB_i} = 35$ mm, $l_1 = l_2 = 100$ mm

1) Parameters

	Mass(kg)	Inertia(kg mm$^{2)}$	CG Position(m) /from CS1	CS(m) /from World
Ground1	-	-	-	[0 0.15 0]
Link1_1	0.053	110.621	[-0.043452 -0.030538 0]	CS1 = [0 0.15 0] CS2 = [-0.081818 0.092497 0]
Link1_2	0.045	90992	[0.040909 -0.028749 0]	CS1 = [-0.081818 0.092497 0] CS2 = [0 0.035 0]
Ground2	-	-	-	[-0.1299 -0.075 0]
Link2_1	0.053	110.621	[0.048173 -0.022361 0]	CS1 = [-0.1299 -0.075 0] CS2 = [-0.039196 -0.117104 0]
Link2_2	0.045	90.992	[0.0044425 0.049802 0]	CS1 = [-0.039196 -0.117104 0] CS2 = [-0.03011 -0.0175 0]
Ground3	-	-	-	[0.129902 -0.074997 0]
Link3_1	0.053	110.621	[-0.0047204 0.0528997 0]	CS1 = [0.129902 -0.074997 0] CS2 = [0.121013 0.024607 0]
Link3_2	0.045	90.992	[-0.0453515 -0.0210535 0	CS1 = [0.121013 0.024607 0] CS2 = [0.030311 -0.0175 0]
Mo-Pl	0.081	46.643	[0 0 0] /from World	CS1 = [0 0.035 0] CS2 = [-0.03011 -0.0175 0] CS3 = [0.030311 -0.0175 0]

2) Inverse dynamics simulation

- Make the 3-DOF parallel robot and move the end-effector (P) by 50mm along the X and Y axes for 2 sec, respectively. The three actuated joint torques can be obtained from the inverse dynamics simulation.

<Trajectory Function>

```
function xd = fcn(dx, tsec, tout)

MAX_CH=3;             % Two Actuators %


% Initializing Variables %
xd=zeros(MAX_CH,1);
x0=[0,0,0]';

%tout=index*dt;

    [c] = coeff_3rd(x0, zeros(MAX_CH,1), x0+dx, zeros(MAX_CH,1), tsec);
    for ch=1:MAX_CH
        xd(ch,1)=c(ch,1)+c(ch,2)*tout+c(ch,3)*tout^2+c(ch,4)*tout^3;
    end

function [c] = coeff_3rd(sp, sv, ep, ev, tsec)
MAX_CH=3;
c=zeros(MAX_CH,4);
for ch=1:MAX_CH
    c(ch,1)=sp(ch,1);
    c(ch,2)=sv(ch,1);
    c(ch,3)=(3.0*(ep(ch,1)-sp(ch,1))-(2.0*sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec);
    c(ch,4)=(-2.0*(ep(ch,1)-sp(ch,1))+(sv(ch,1)+ev(ch,1))*tsec)/(tsec*tsec*tsec);
end
```

<Inverse Kinematics Function>

```
function thd = fcn(xd)

persistent index thd_p

% Initializing Variables %
if isempty(index), index=0; end
if isempty(thd_p), thd_p=(pi/180)*[-144.9004,-24.9004,95.0996]'; end

% Kinematic Parameters %
ra=150; rb=35; L1=100; L2=100;

pos=xd(1:2,1); ang=xd(3,1);

w_index=1;    % In workspace %

% Vertex Points %
A=ra*[0,-sqrt(3)/2, +sqrt(3)/2; 1, -1/2, -1/2];
B=rb*[0,-sqrt(3)/2, +sqrt(3)/2; 1, -1/2, -1/2];
```

```
th=zeros(3,1);
R=[cos(ang), -sin(ang); sin(ang), cos(ang)];
for i=1:3
    d=pos+R*B(:,i)-A(:,i);
    e1=-2*L1*d(2,1); e2=-2*L1*d(1,1);
    e3=d(1,1)^2+d(2,1)^2+L1^2-L2^2;
    temp=e1^2+e2^2-e3^2;
    if temp>=0
        th(i,1)=2*atan((-e1-sqrt(temp))/(e3-e2));
    else
        th(i,1)=0;
        w_index=-1;
        %disp('Out of Workspace');
    end
end

% Result %
if w_index==1
    thd=[th(1,1), th(2,1), th(3,1)]';
else
    thd=thd_p;
end

% Save the current value %
thd_p=thd;
```

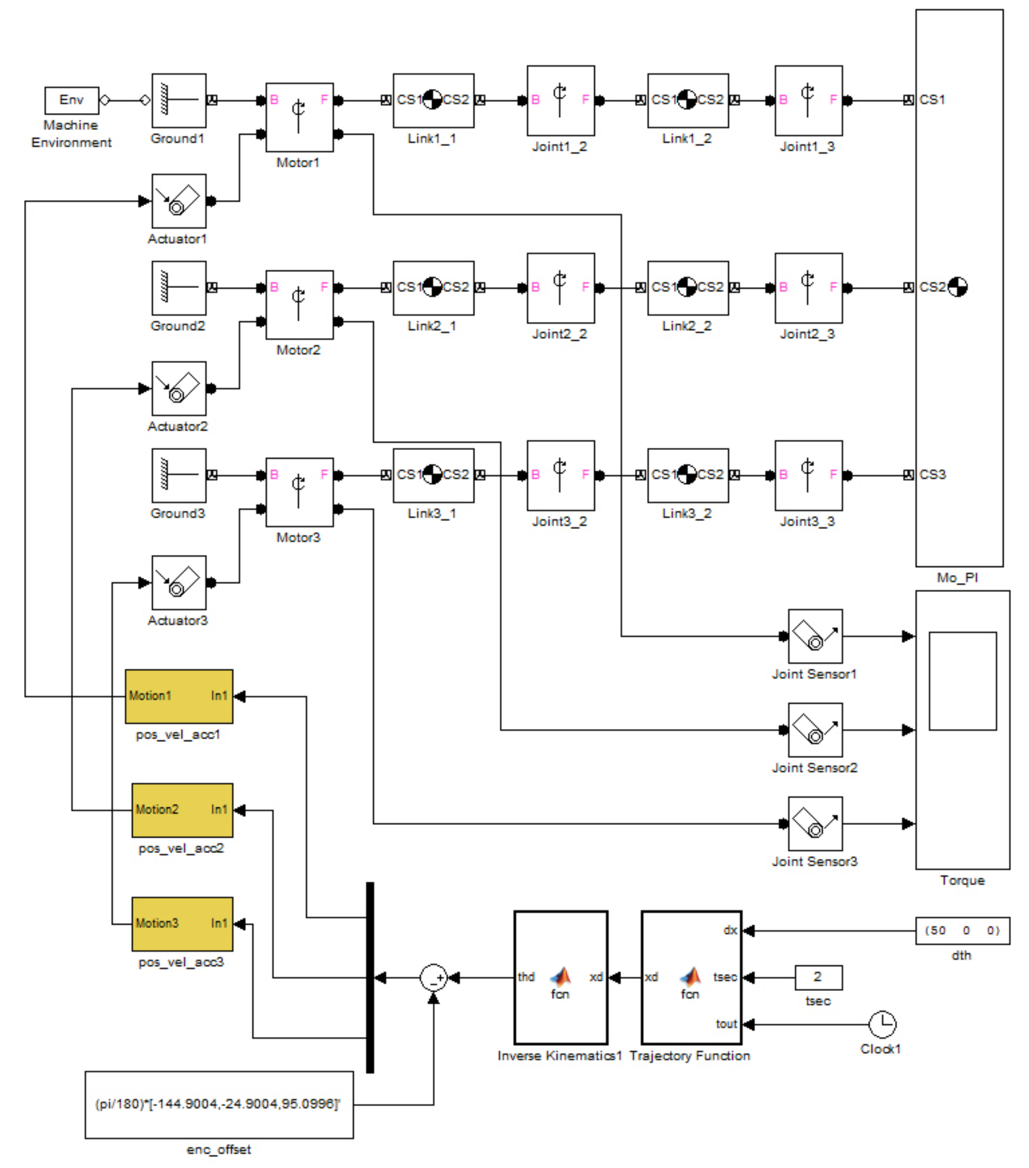
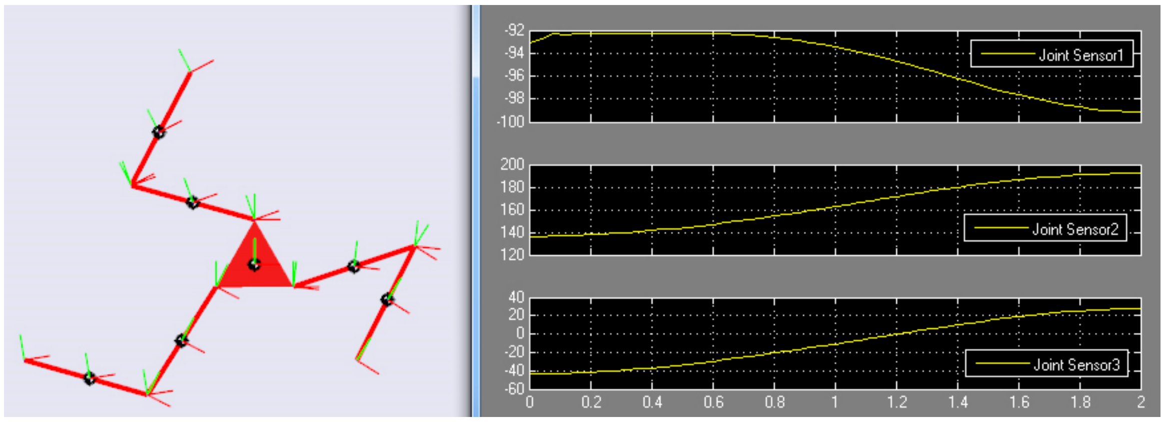


Fig.34: Planar 3-DOF parallel robot model.



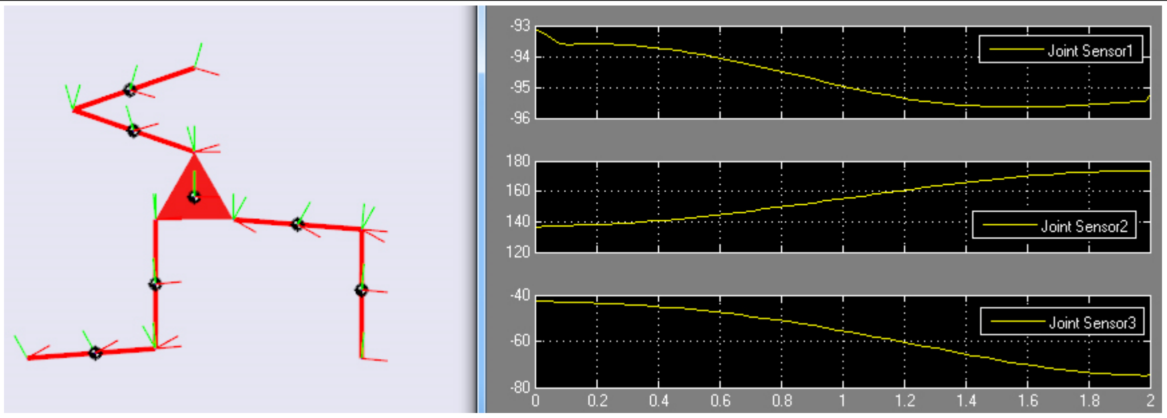
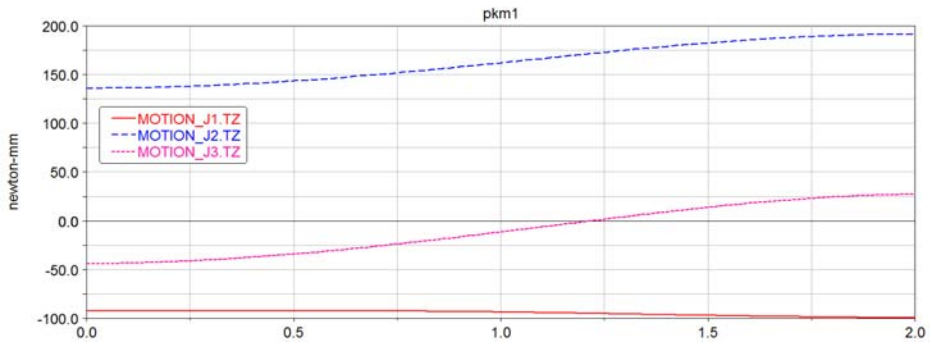
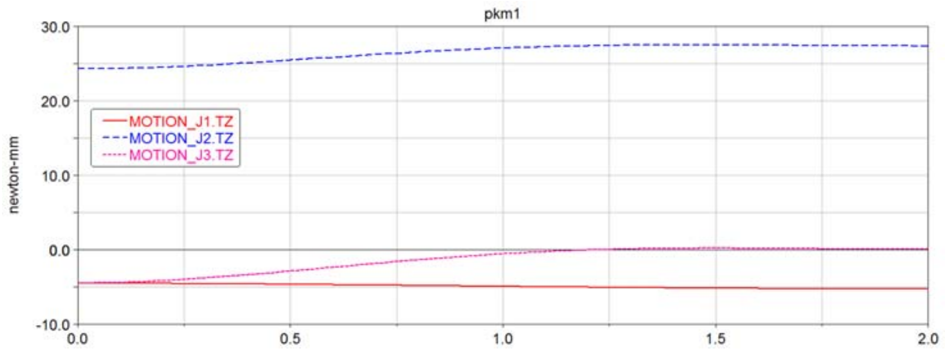


Fig. 35: Simulation results of planar 3-DOF parallel robot model.

[ADAMS simulation result (move along the X axis)]



[ADAMS simulation result (move along the Y axis)]


pkm1
newton-mm
MOTION_J1.TZ
MOTION_J2.TZ
MOTION_J3.TZ