UNIVERSITÀ
DEGLI STUDI
FIRENZE

Trajectory Tracking and Pick&Place Tasks with Arduino Tinkerkit Braccio

Angelo D'Amante
Lorenzo Falai

Engineering School - Automation Laboratory

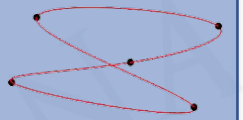
Academic Year 2021/2022

Introduction

Platform
DH Table
Workspace

Modeling

Trajectory Tracking



Trajectory Generation
Simulink VR Toolbox
Matlab Robotics Toolbox

Pick&Place

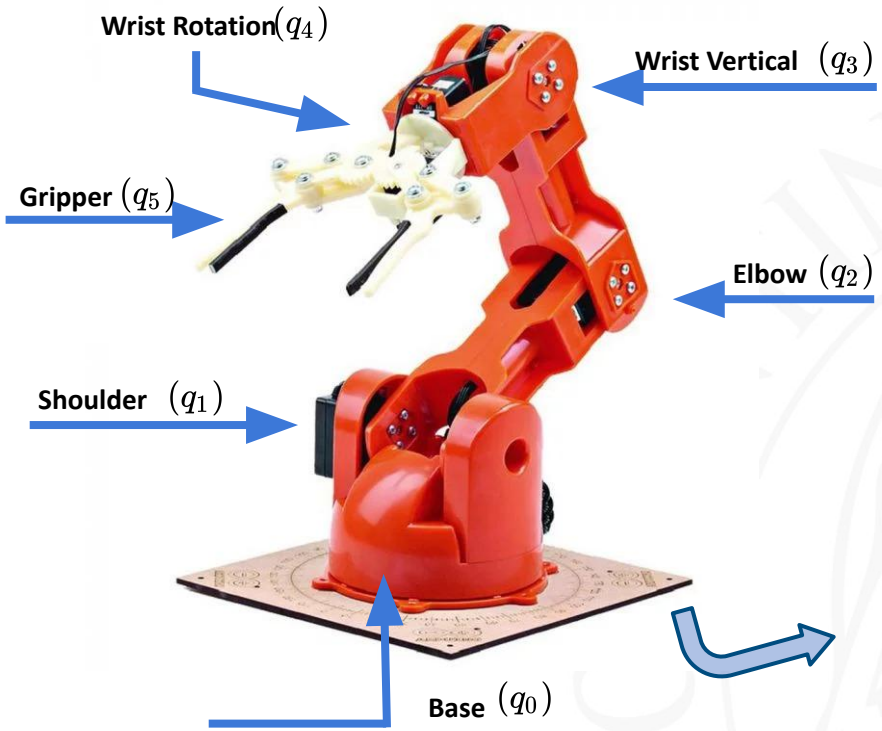
Our Kin Ikin Solver
Stateflow
Hardware Issues

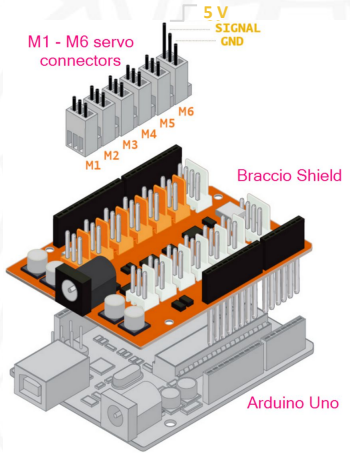
Further Improvements

Modeling

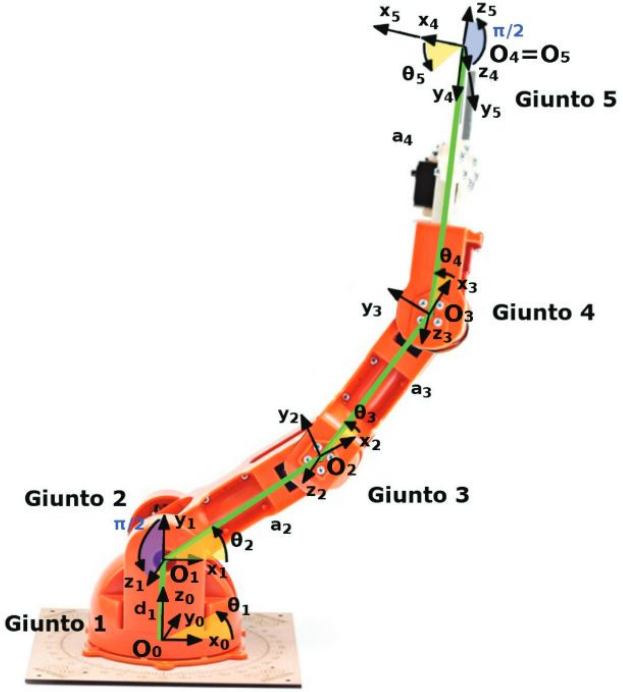
- Arduino Tinkerkit Braccio
- DH Parameters
- Workspace

Arduino Tinkerkit Braccio



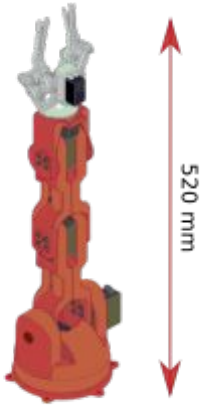


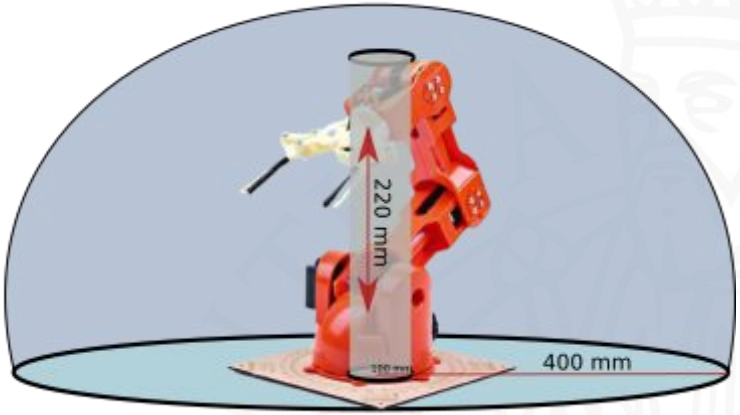
Denavit Hartenberg Parameters



Link	a_i	α_i	d_i	ϑ_i
1-Base	0	$\frac{\pi}{2}$	d_1	ϑ_1
2-Forearm	a_2	0	0	ϑ_2
3-Arm	a_3	0	0	ϑ_3
4-Hand	a_4	0	0	ϑ_4
5-EE	0	$\frac{\pi}{2}$	0	ϑ_5

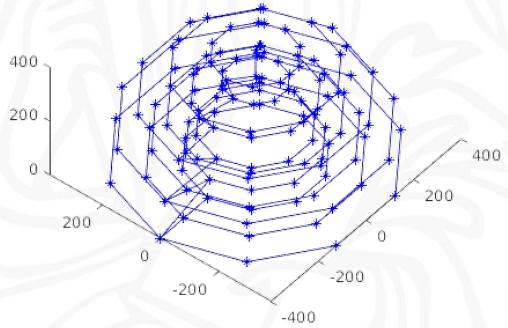
Workspace






```
function
      makeWS(dRadius,dInnerRadius,dInnerHeight
      )
      return oWorkspace;
end
```

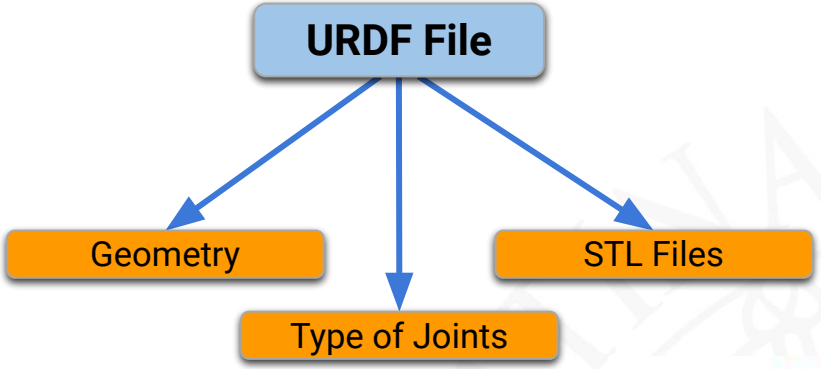
```
makeWS(400, 100,
      220)
```

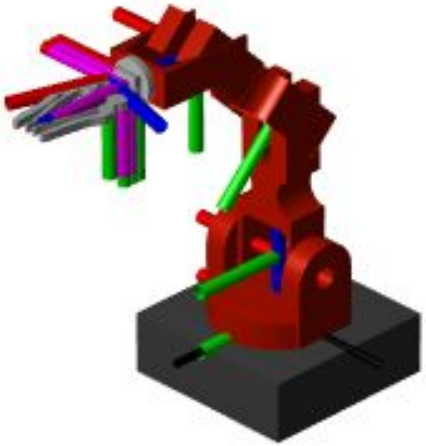


Trajectory Tracking

- Matlab Simulation
  - Braccio Model
  - Robotics Toolbox
- Simulink Simulation
  - Virtual Reality Toolbox
- Hardware Issues

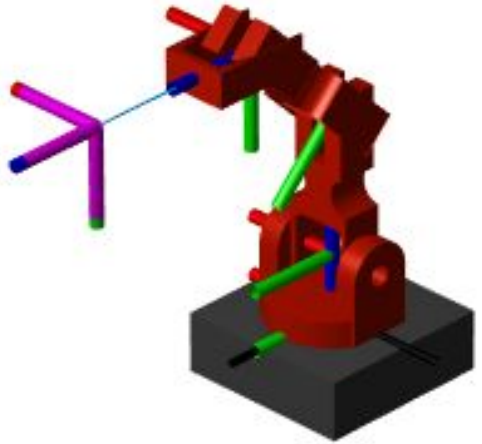
Matlab Simulation - Braccio Model





Braccio: 6 Joints
[q0 q1 q2 q3 q4 q5]

Reduce Number of Joints



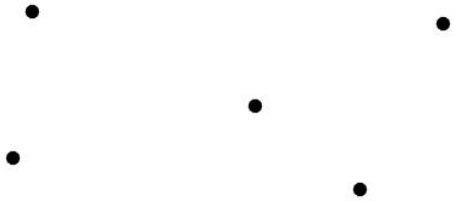
Braccio: 4 Joints
[q0 q1 q2 q3]

Matlab Simulation - Trajectory Generation

1. Define a Set of Waypoints

code

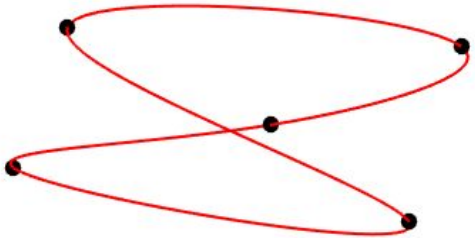
```
wayPoints=[initial_point; ...
           (rand*0.15)+0.05 ((rand*4)-2)/10 (rand*0.35)+0.05; ...
           (rand*0.15)+0.05 ((rand*4)-2)/10 (rand*0.35)+0.05; ...
           (rand*0.15)+0.05 ((rand*4)-2)/10 (rand*0.35)+0.05; ...
           (rand*0.15)+0.05 ((rand*4)-2)/10 (rand*0.35)+0.05; ...
           initial_point];
```



2. Generate a Trajectory

code

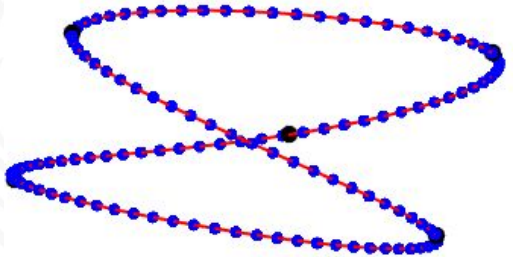
```
traj=cscvn(wayPoints');
fnplt(traj,'r',2);
```



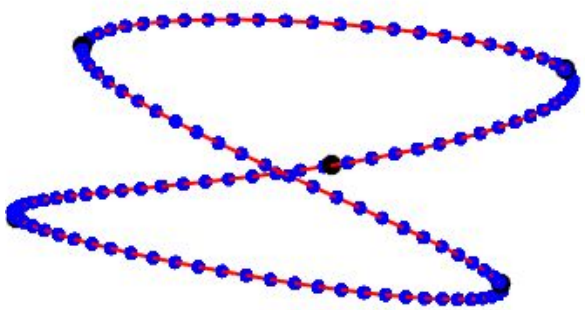
3. Generate a Finite Number of Points on the Trajectory

code

```
[n,~]  =size(wayPoints);
totalPoints=n*20;
x=linspace(0,traj.breaks(end),totalPoints);
eePos=ppval(traj,x);
```



Matlab Simulation - Inverse Kinematic Solver

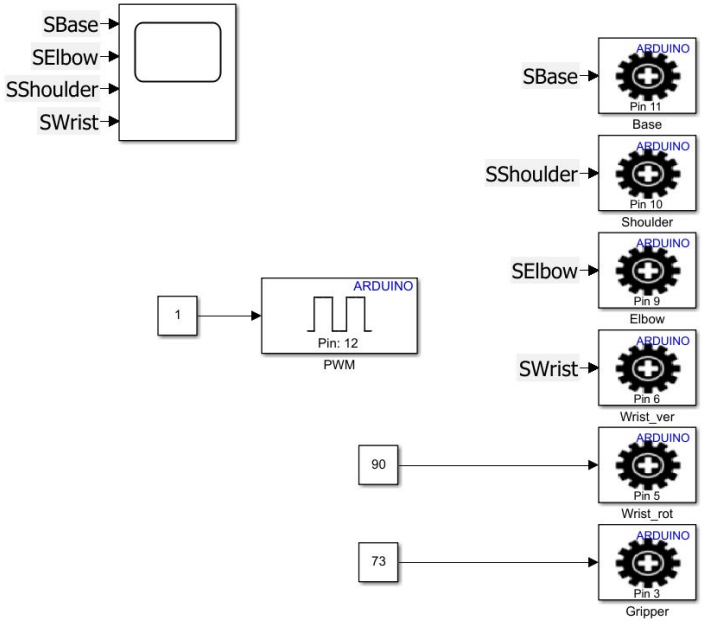


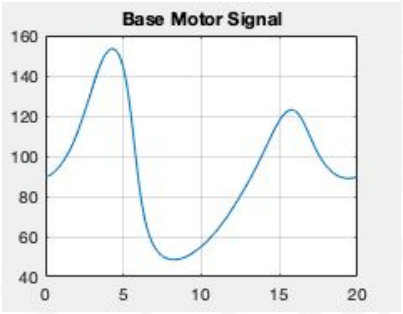
```
ik = robotics.InverseKinematics( 'RigidBodyTree' ,robot);
ik.SolverAlgorithm =  'LevenbergMarquardt' ;
weights = [0 0 0 1 1 1];
initialguess = config;

for idx = 1:size(eePos,2)
    tform = trvec2tform(eePos(:,idx)');
    configSoln(idx,:) = ik( 'end_effector' ,tform,weights,initialguess);
    initialguess = configSoln(idx,:);
end
```

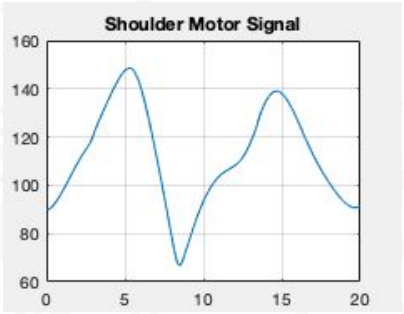
q_{01}	q_{11}	q_{21}	q_{31}
q_{02}	q_{12}	q_{22}	q_{32}
…	…	…	…

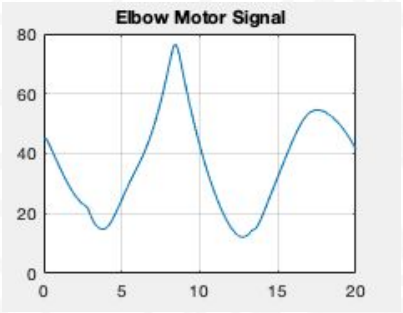
JOINT CONFIGURATION MATRIX

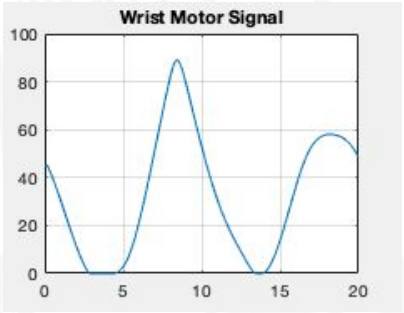
Trajectory Tracking - Hardware Implementation



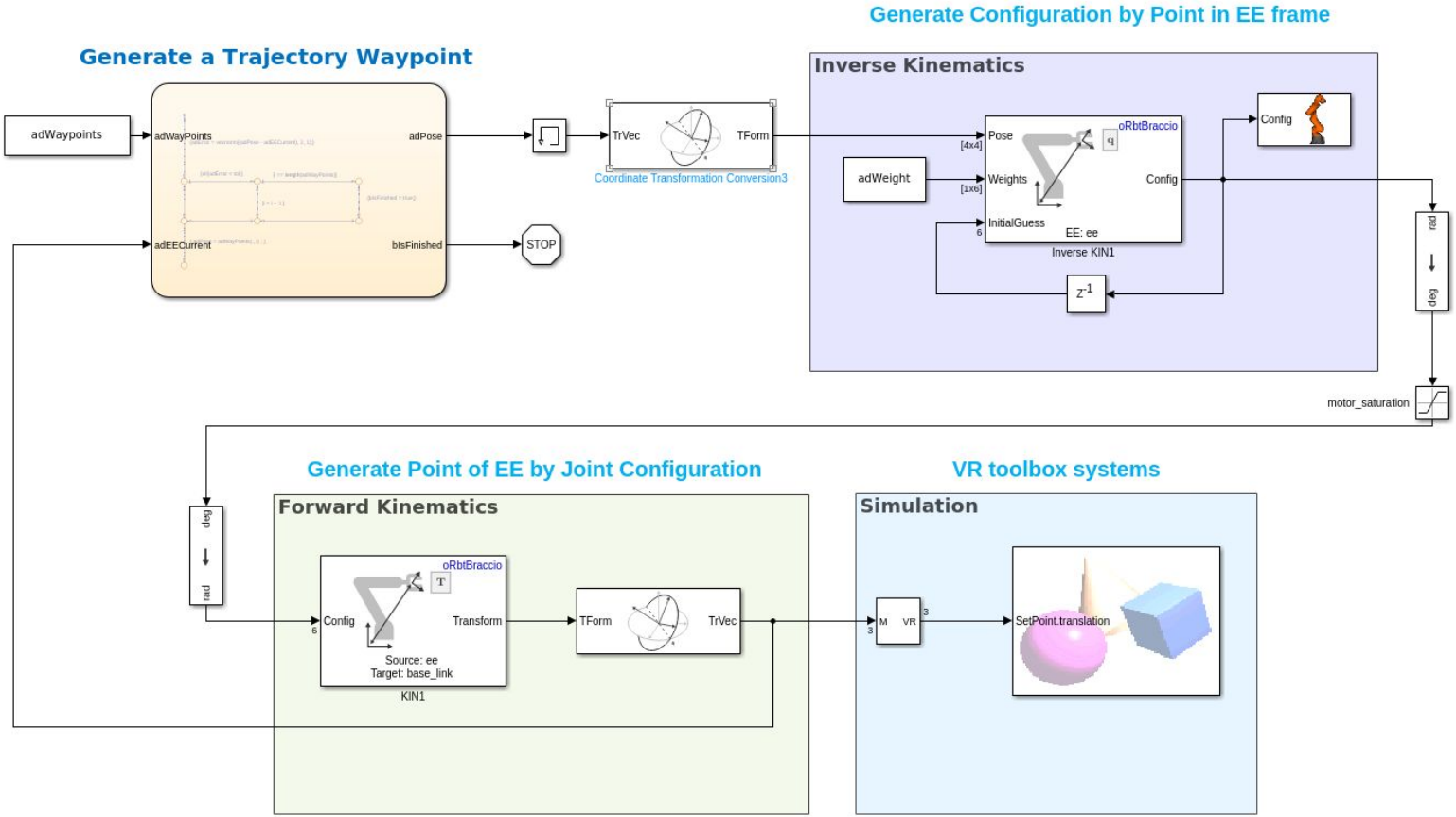




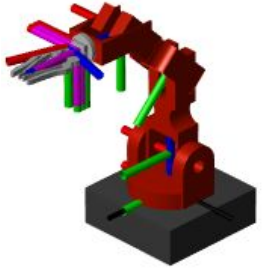




Simulink Simulation - Model



Hardware Implementation - Issues



?

Issues

1. Blocks from the Matlab Robotics Toolbox don't support code generation
2. The function that generates the trajectory can't be used in external mode
3. How do we read the joint configuration matrix?

HW Evaluation

Flash Memory: 32 KB
EEPROM: 1 KB
SRAM: 2 KB

Flash Memory: 256 KB
EEPROM: 4 KB
SRAM: 8 KB

`evaluateUsageMemory.m` Script

Pick&Place

- KIN/IKIN Algorithms
- KIN/IKIN Implementation
  - Simulink Blocks
  - System Objects
  - Naive-Approach Model
  - Solutions Comparison
- Testing
- Pick&Place Model

Algorithm - Forward Kinematics

```
1: procedure KIN(adD, adQ, adA, adAlpha, adOffset)
2:     adJC = adQ + adOffset
3:     adT = eye(4,4)
4:     for i in [1, nDOF ] do
5:         d = adD[i], q = adJC[i], a = adA[i], alpha = adAlpha[i]
6:         adT = adT * functions.computeT(d, q, a, alpha)
7:     end for
8:     return adT
9: end procedure
```

```
function computeT(dD, dQ, dA, dAlpha)
       if !isNum(dD, dQ, dA, dAlpha) return eye(4,4);

       adT = [       cos(dQ)      -sin(dQ)*cos(dB)     sin(dQ)*sin(dB)      dA*cos(dQ); ...
                     sin(dQ)      cos(dQ)*cos(dB)      -cos(dQ)*sin(dB)     dA*sin(dQ); ...
                     0            sin(dB)              cos(dB)              dD;         ...
                     0            0                    0                    1
       ];
       return adT;
end
```

Algorithm - Inverse Kinematics

```
1: procedure IKIN(adPoint, adWS)
2:     x = adPoint(1), y = adPoint(2), z = adPoint(3)
3:     adJC initialization with Home Config
4:     If (adPoint ∉ workspace) then return [false, adJC]
5:     bFlag, base = computeBaseAngle(x, y)
6:     radius = √(x² + y²)
7:     If !bFlag then radius = -radius
8:     z = z - legthOfBaseLink
9:     for φ = −2π → 2π  do
10:        bFlag, shoulder, elbow, wrist = solvePlanar(radius, z, φ)
11:        If bFlag then break
12:    end for
13:    adJC = [base, shoulder, elbow, wrist]
14:    return [bFlag, adJC]
15: end procedure
```

```
function computeBaseAngle(x, y)
        q0      = atan2(y, x);
        rangeCheckForBaseJoint(q0);
        return [flag, q0];
end
```

q_0

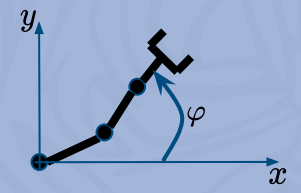
```
function solvePlanar(x, y, phi)
        Compute q1, q2, q3;
        rangeCheckForJoints(q1, q2, q3);
        return [flag, q1, q2, q3];
end
```

q_1

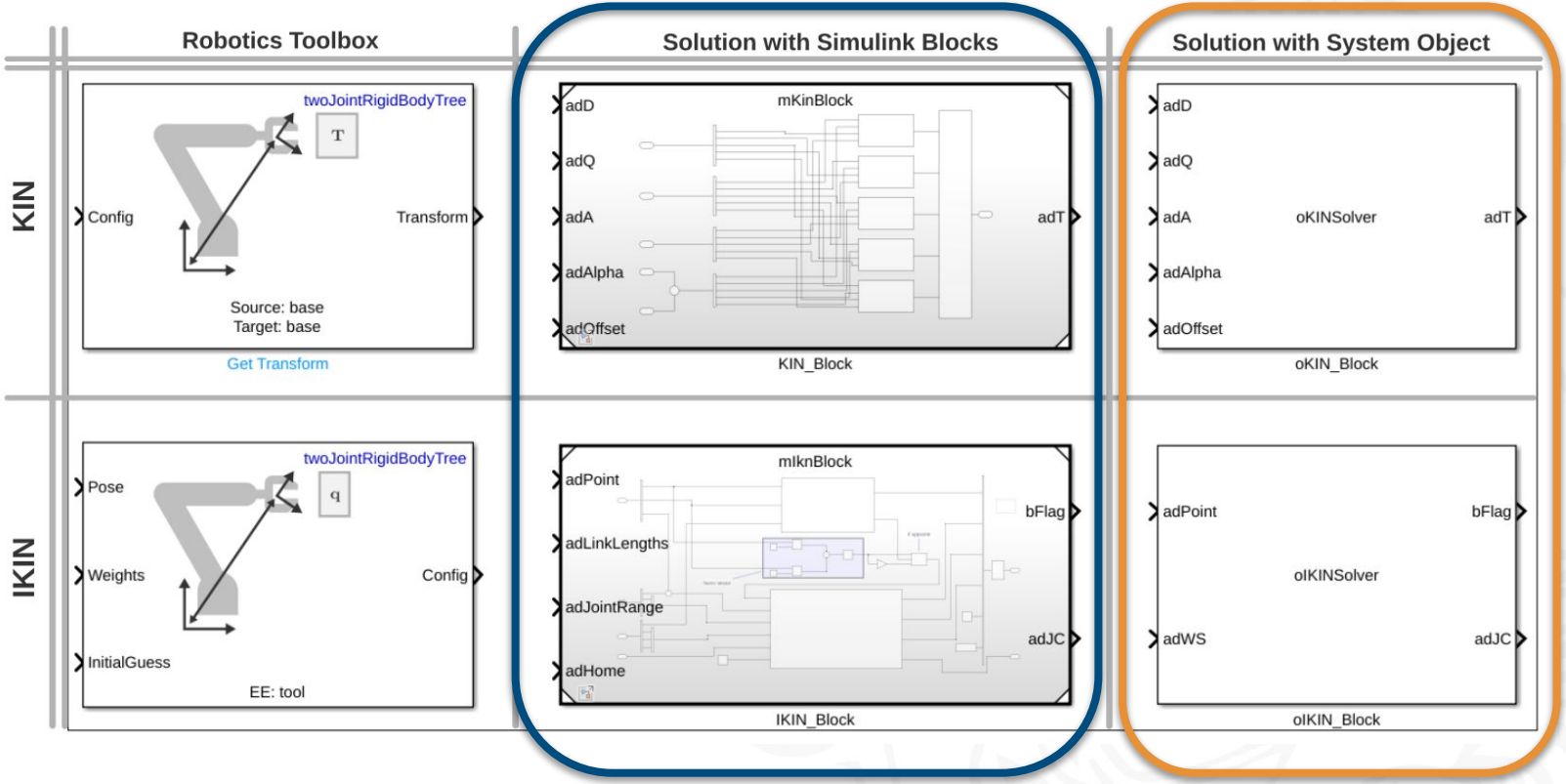
q_2

q_3

Loop the angle until the flag results true

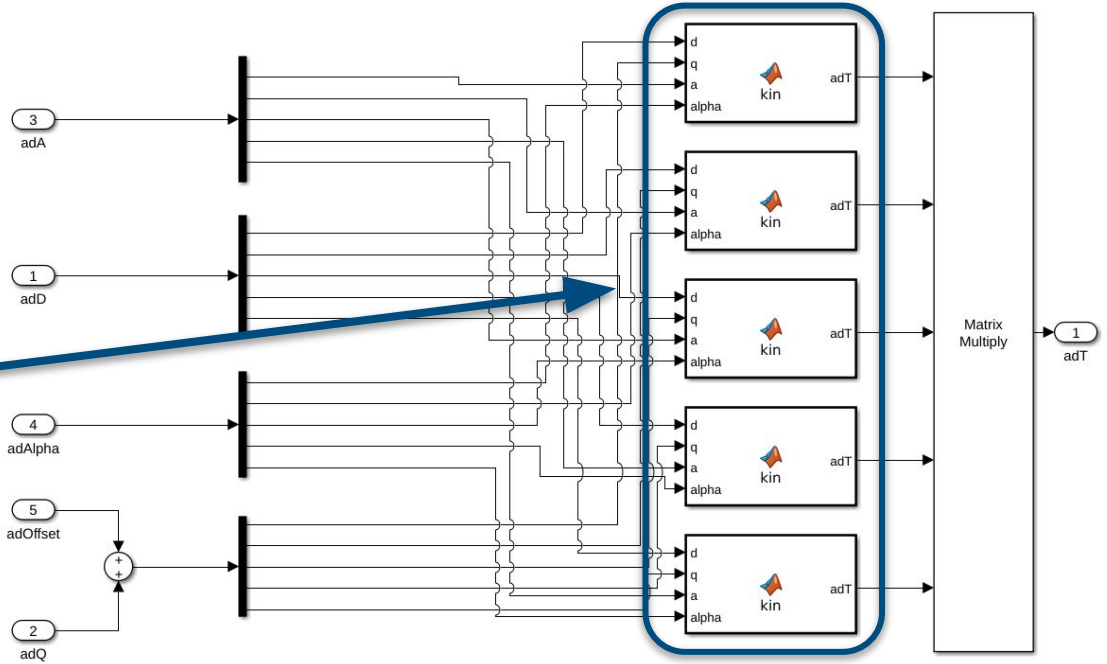


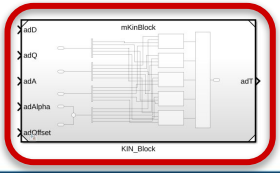
KIN/IKIN Implementation

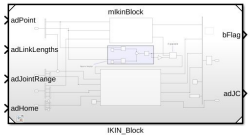


Implementation - Simulink Blocks - KIN

```
1: procedure KIN(adD, adQ, adA, adAlpha, adOffset)
2:     adJC = adQ + adOffset
3:     adT = eye(4,4)
4:     for i in [1, nDOF] do
5:         d = adD[i], q = adJC[i], a = adA[i], alpha = adAlpha[i]
6:         adT = adT * functions.computeT(d, q, a, alpha)
7:     end for
8:     return adT
9: end procedure
```



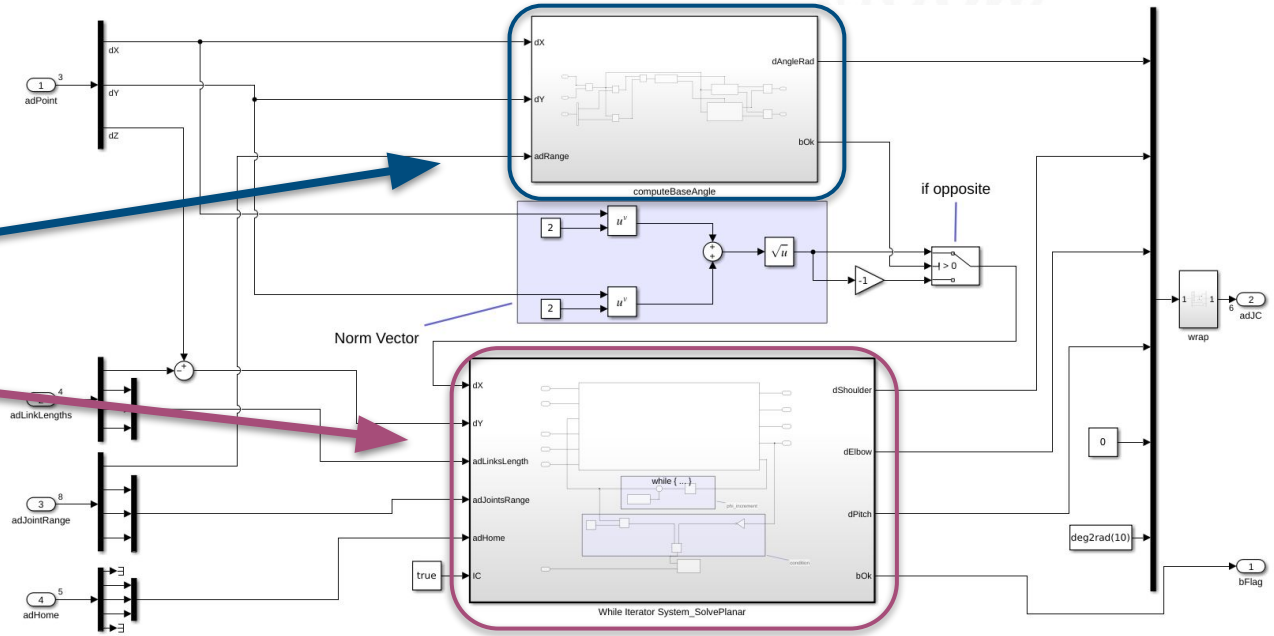


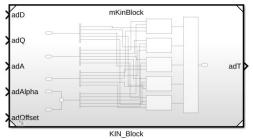


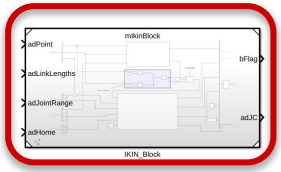

```
function adT = kin(d, q, a, alpha)
    adT = functions.computeT(d, q, a, alpha);
end
```

Implementation - Simulink Blocks - IKIN

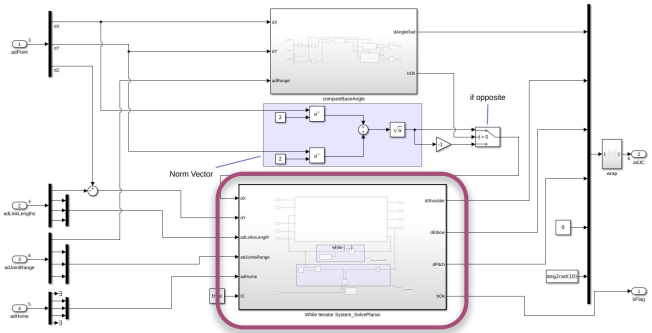
```
1:  procedure IKIN(adPoint, adWS)
2:      x = adPoint(1), y = adPoint(2), z = adPoint(3)
3:      adJC initialization with Home Config
4:      If (adPoint ∉ workspace) then return [false, adJC]
5:      bFlag, base = computeBaseAngle(x, y)
6:      radius = √(x² + y²)
7:      If !bFlag then radius = -radius
8:      z = z - legthOfBaseLink
9:      for φ = −2π → 2π  do
10:         bFlag, shoulder, elbow, wrist = solvePlanar(radius, z, φ)
11:         If bFlag then break
12:     end for
13:     adJC = [base, shoulder, elbow, wrist]
14:     return [bFlag, adJC]
15: end procedure
```



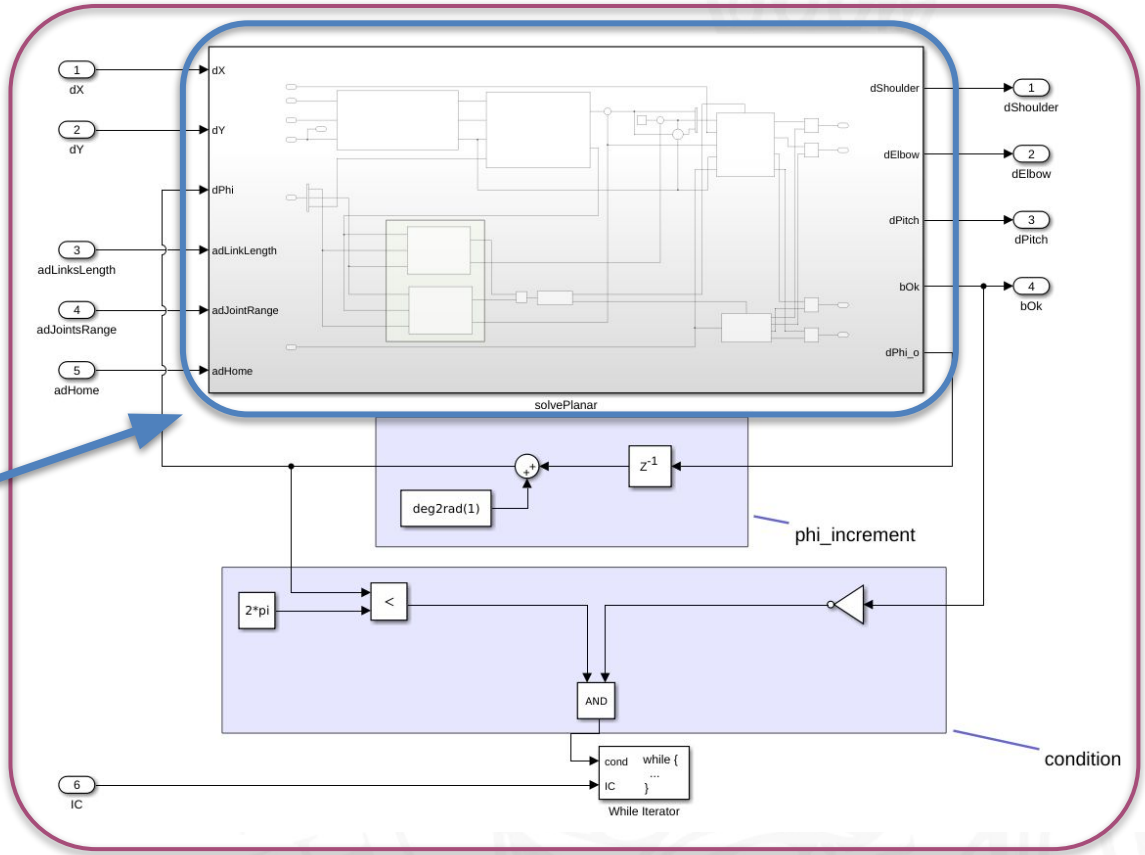




Implementation - Simulink Blocks - IKIN




```
9:   for φ = −2π → 2π do
10:      bFlag, shoulder, elbow, wrist = solvePlanar(radius, z, φ)
11:      If bFlag then break
12:  end for
```



Implementation - System Objects

Modularity Software with Open/Closed Principle

Joint

- minAngleRad: double
- maxAngleRad: double

+ Joint(angleDown, angleUp)
+ isInRange(angle) : bFlag

Robot5Dof

// JOINTS
+ ojBase, + ojShoulder, + ojElbow, + ojPitch, + ojWrist

// LINKS
+ olGround, + olArm, + olForearm, + olHand

// Properties
+ adHome, + nDOF

+ Robot5Dof(aoJoints, aoLinks, adHome)
+ setHomeConf(adHome)
+ KIN(dhParam) : T05
+ IKIN(adPoint) : [flag, Q]
- computeBase(z) : angle
- solvePlanar(x, y, phi) : [flag, sh, el, pitch]

Link

- length: double

+ Link(length)
+ getLength( ): double

```
% Joints Definition %
oJointBase      = classes.Joint(deg2rad(0), deg2rad(180));
oJointShoulder  = classes.Joint(deg2rad(15), deg2rad(165));
oJointElbow     = classes.Joint(deg2rad(0), deg2rad(180));
oJointPitch     = classes.Joint(deg2rad(0), deg2rad(180));
oJointWrist     = classes.Joint(deg2rad(0), deg2rad(180));

% Make Array of Joints %
aoJoints     = [ ... ];

% Links Definition %
oLinkGround     = classes.Link(70);
oLinkArm        = classes.Link(125);
oLinkForearm    = classes.Link(125);
oLinkHand       = classes.Link(195);

% Make Array of Links %
aoLinks         = [ ... ];

% Create Robot Object %
oBraccio     = Robot5Dof(aoLinks, aoJoints, adHome);

% KIN/IKIN Algorithms %
adT                  = oBraccio.KIN(D, Q, A, Alpha, Offset);
[bFlag, adJC]    = oBraccio.IKIN(adPoint, adWS);
```

How can we use these objects in Simulink?

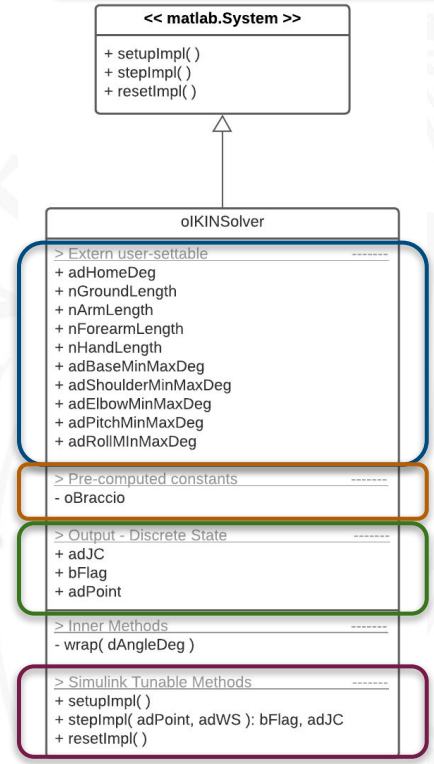
Memory Issues with matlab function block

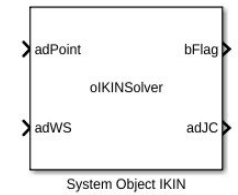
Implementation - System Objects

How can we use these objects with simulink?

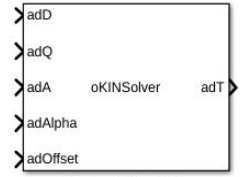
System Objects works with simulink

```
classdef objectSystems < matlab.System
    properties(Access = public)
      % User-settable constants %
    end
    properties(Access = private)
      % Pre-computed constants %
    end
    properties(DiscreteState)
      % Output State - tunable %
    end

    methods(Access = protected)
      function setupImpl(obj, inputs)
              % Perform one-time calculations %
              % sets up a System object %
      end
      function [outputs] = stepImpl(obj, inputs)
              % specifies the algorithm to execute at run-time %
              % updates the object's state values using the inputs %
          end
          function resetImpl(obj)
              % reset the states to a set of initial values %
          end
    end
end
```





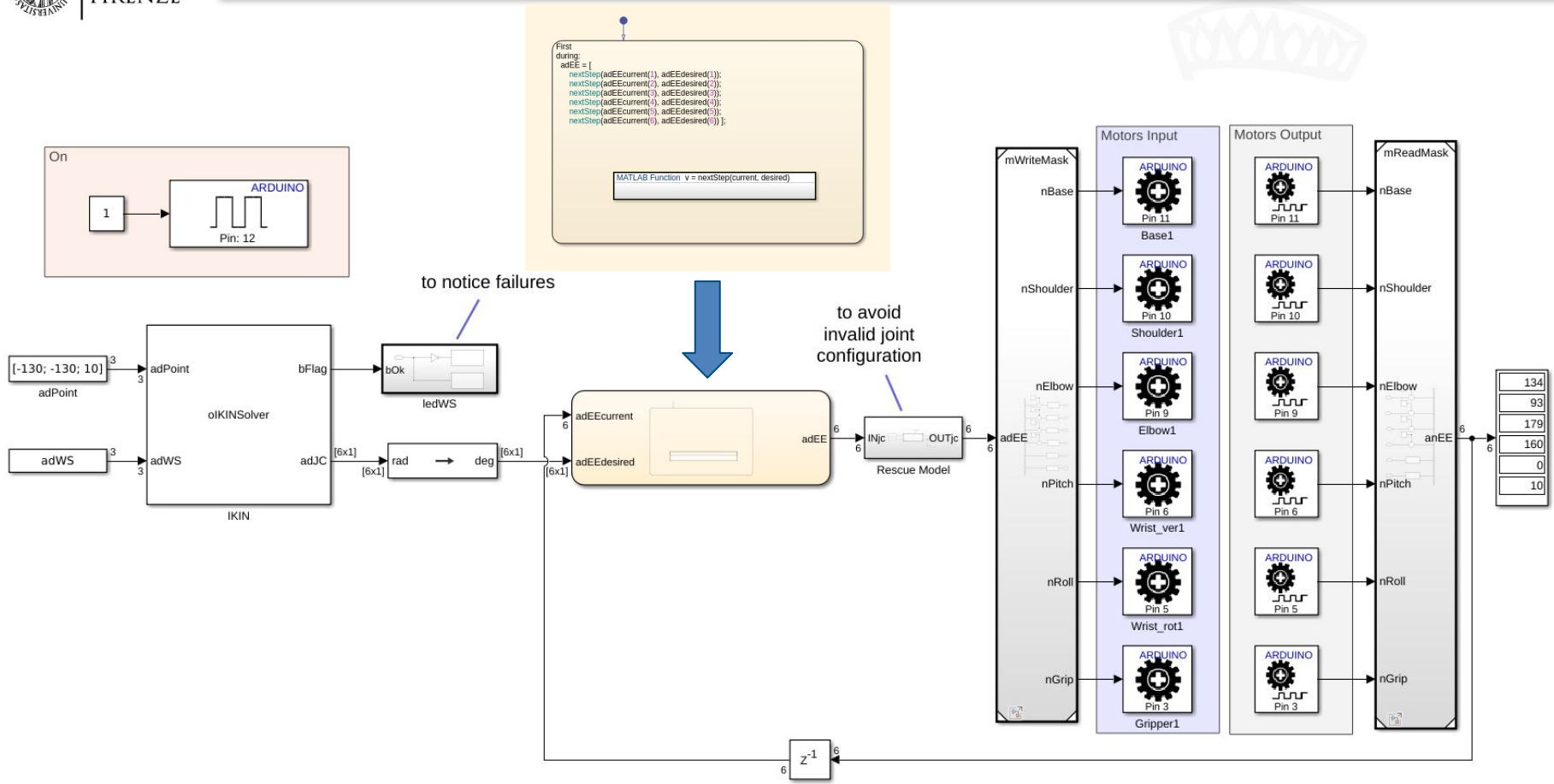
System Object IKIN



System Object KIN

Matlab System Blocks

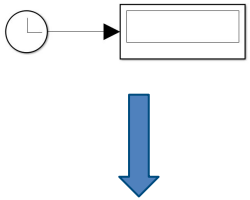
Naive Approach



Implementation - Comparison

Simulink Blocks

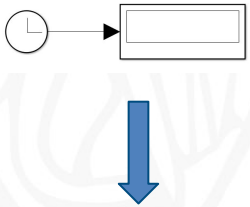
- Storage:
  - Program 35380 B
  - Data 2994 B
- Timing:
  - Build 15.849 sec
  - Sim Time 1.76



Specific Purpose

System Objects

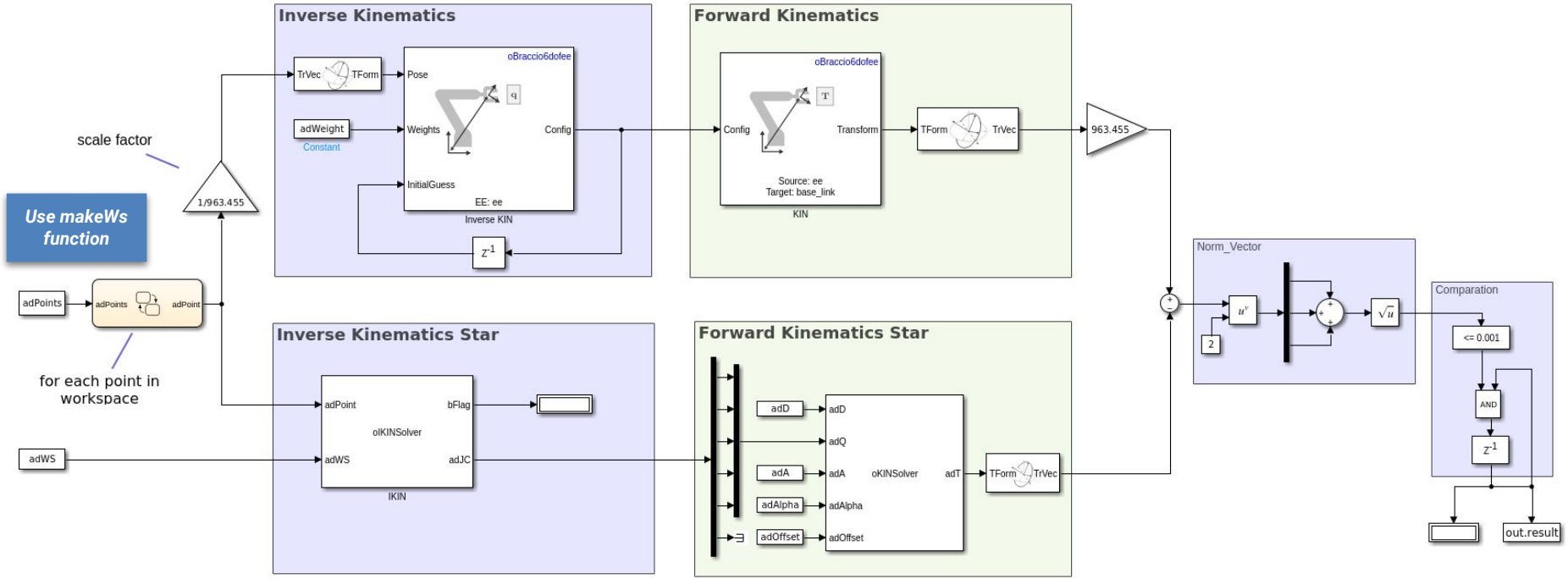
- Storage:
  - Program 39048 B
  - Data 3322 B
- Timing:
  - Build 37.771 sec
  - Sim Time 1.78



Like Generic Purpose

Implementation - Evaluation

Verify that our solution is equivalent to the toolbox



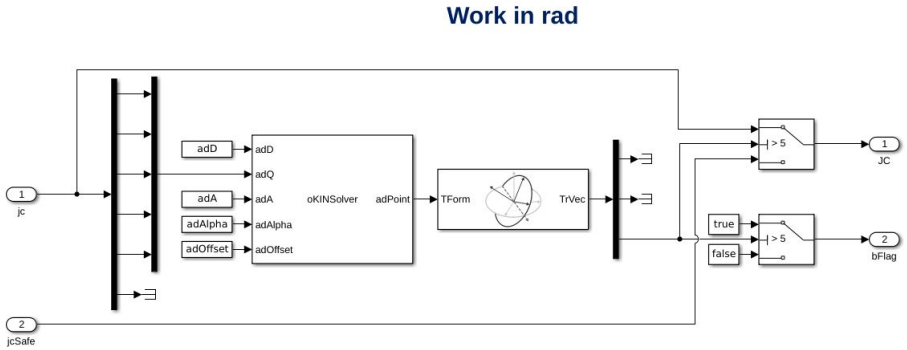
Testing

UnitTest for each algorithm and module:

`initHW.m Script` →

```
Ut: PASS ---- carnotRule
Ut: PASS ---- computeT
UT: PASS ---- Robot built in: 1.73 ms
UT: PASS ---- IK computed in: 1.98 ms
UT: PASS ---- FK computed in: 0.01 ms
Ut: PASS ---- FK_IK class works fine
Ut: PASS ---- Probably good for arduino Uno
Ut: PASS ---- Probably good for arduino Mega
```

Rescue Model to avoid invalid joints configuration:

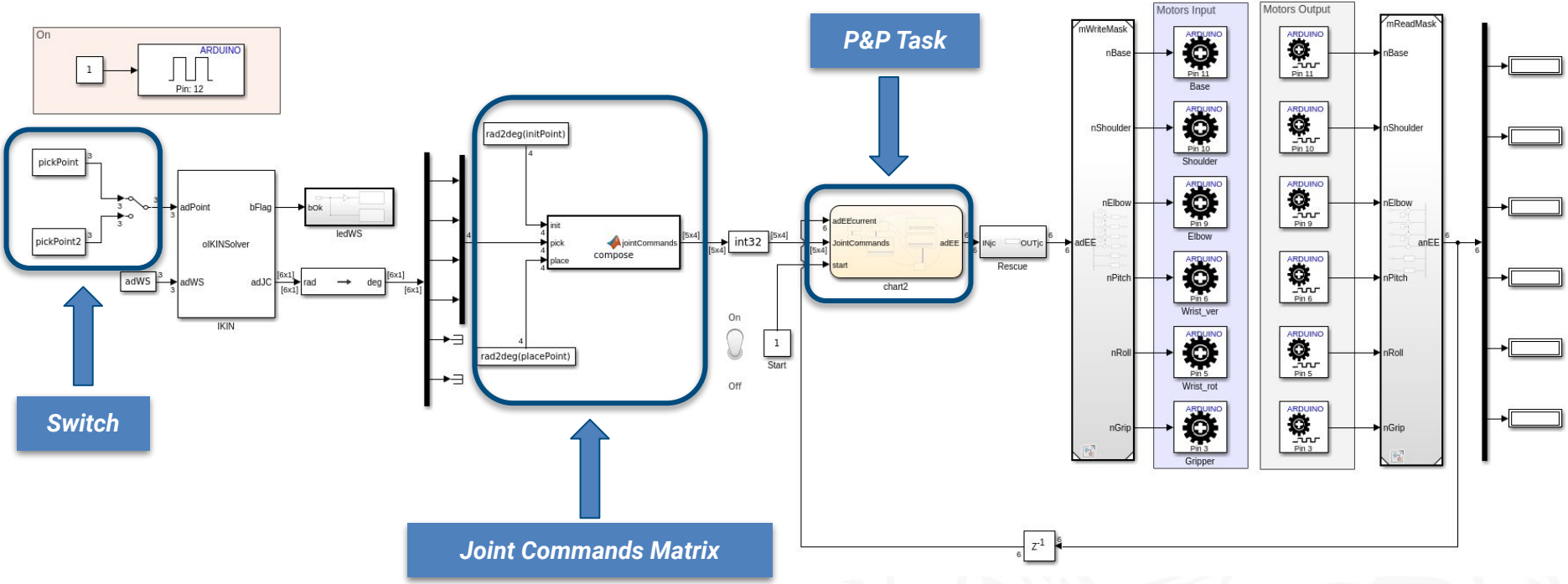


Use Forward Kinematics to evaluate z coordinate

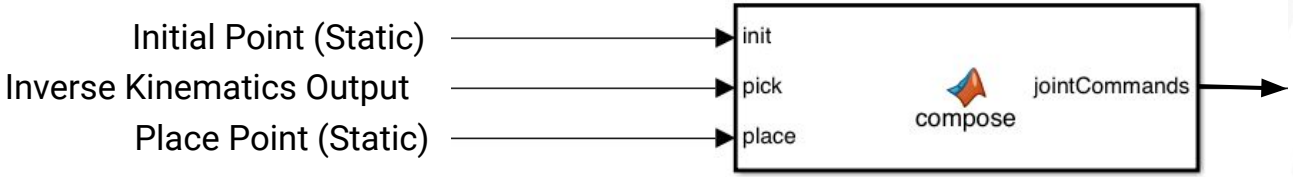
↓

z under threshold: Braccio in Home Configuration

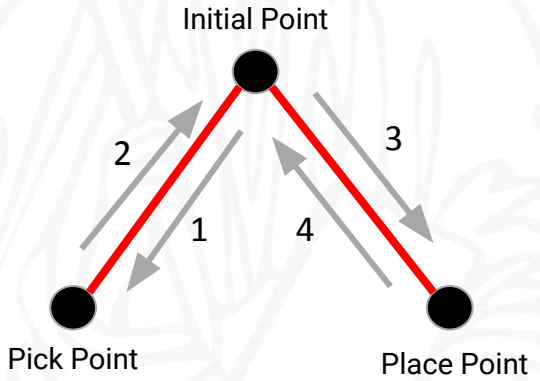
P&P Model


On
ARDUINO
Pin: 12
pickPoint
pickPoint2
adWS
adPoint
adWS
bFlag
adJC
oIKINSolver
IKIN
bOk
ledWS
rad
deg
rad2deg(initPoint)
rad2deg(placePoint)
init
pick
place
jointCommands
compose
int32
On
Off
Start
adEEcurrent
JointCommands
start
adEE
chart2
INjc
OUTjc
Rescue
mWriteMask
nBase
nShoulder
nElbow
adEE
nPitch
nRoll
nGrip
Motors Input
Base
Shoulder
Elbow
Wrist_ver
Wrist_rot
Gripper
Motors Output
mReadMask
anEE
z^{-1}
P&P Task
Switch
Joint Commands Matrix

P&P Model - Trajectory Generation



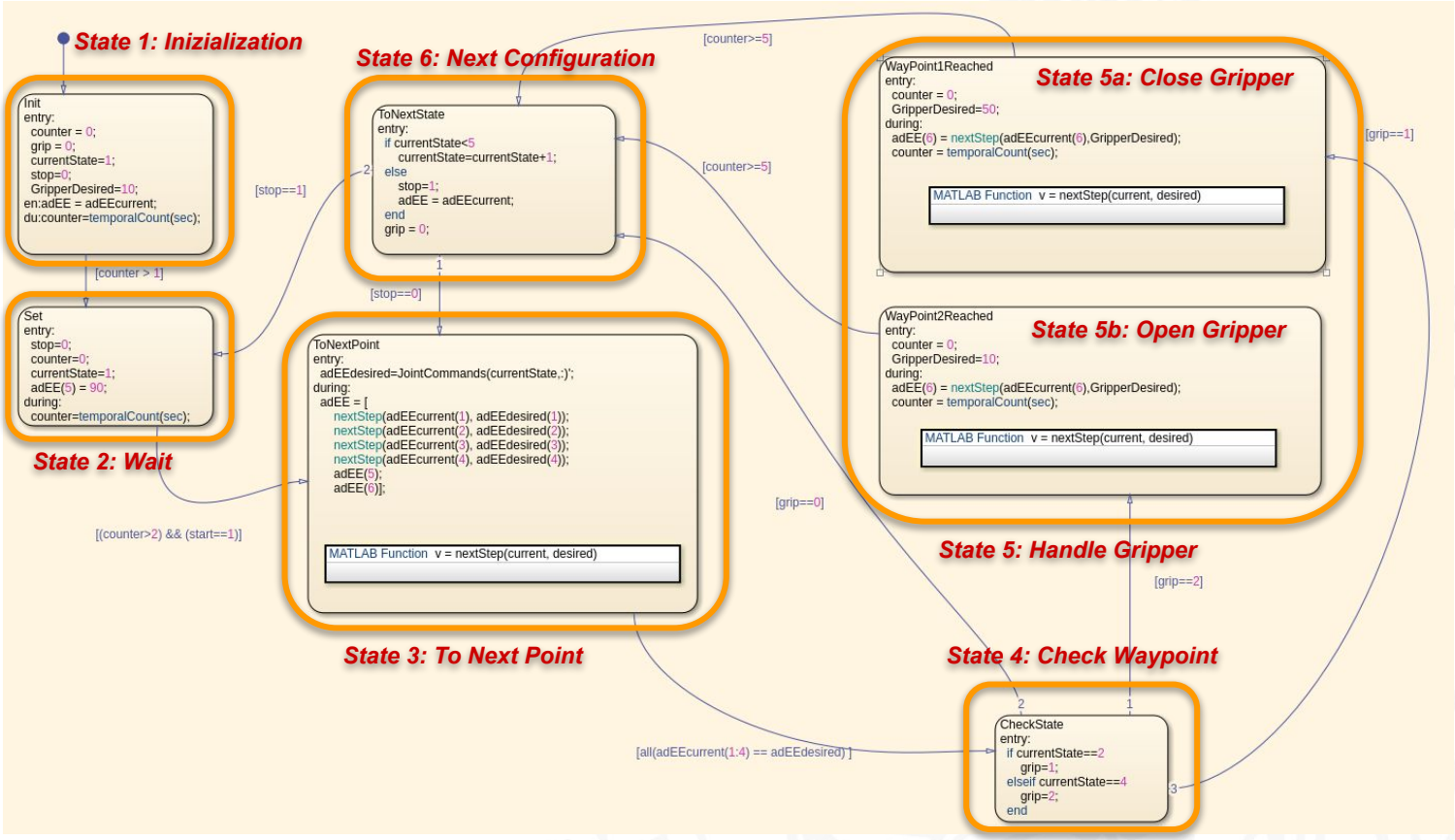
Init	q_0	q_1	q_2	q_3
Pick	q_0	q_1	q_2	q_3
Init	q_0	q_1	q_2	q_3
Place	q_0	q_1	q_2	q_3
Init	q_0	q_1	q_2	q_3



P&P Model - Chart Stateflow

Input:
- Joint Configuration Matrix
- Current Joint Configuration

Output:
- Joint Configuration Vector

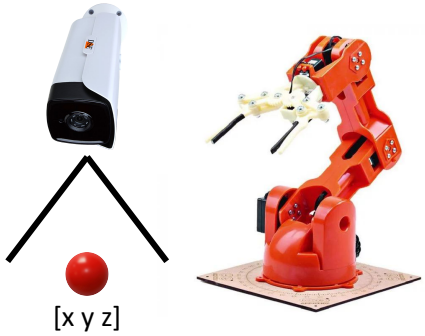


Further Improvements

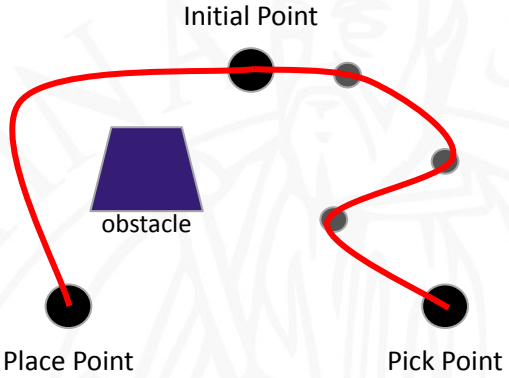
- Results
- Further Improvements

Further Improvements

Using a camera to replace static input points in the workspace with dynamic inputs



Writing a function to generate a trajectory (obstacle avoidance)



Using the camera to also recognize different objects

UNIVERSITÀ DEGLI STUDI FIRENZE

Trajectory Tracking and Pick&Place Tasks with Arduino Tinkerkit Braccio

Angelo D'Amante
Lorenzo Falai

https://github.com/AngeloDamante/arm-manipulator-5dof

Engineering School - Automation Laboratory

Academic Year 2021/2022