Exercise 2b: Model-based control of the ABB IRB 120

Prof. Marco Hutter*

Teaching Assistants: Vassilios Tsounis, Jan Carius, Ruben Grandia†

October 31, 2017

Abstract

In this exercise you will learn how to implement control algorithms focused on model-based control schemes. A MATLAB visualization of the robot arm is provided. You will implement controllers which require a motion reference in the joint-space as well as in the operational-space. Finally, you will learn how to implement a hybrid force and motion operational space controller. The partially implemented MATLAB scripts, as well as the visualizer, are available at `http://www.rsl.ethz.ch/education-students/lectures/robotdynamics.html`.

Figure 1: The ABB IRW 120 robot arm.

---

*original contributors include Michael Blösch, Dario Bellicoso, and Samuel Bachmann
†tsounisv@ethz.ch, jan.carius@mavt.ethz.ch, ruben.grandia@mavt.ethz.ch

1 Introduction

The robot arm and the dynamic properties are shown in Figure 2. The kinematic and dynamic parameters are given and can be loaded using the provided MATLAB scripts. To initialize your workspace, run the `init_workspace.m` script. To start the visualizer, run the `loadviz.m` script.
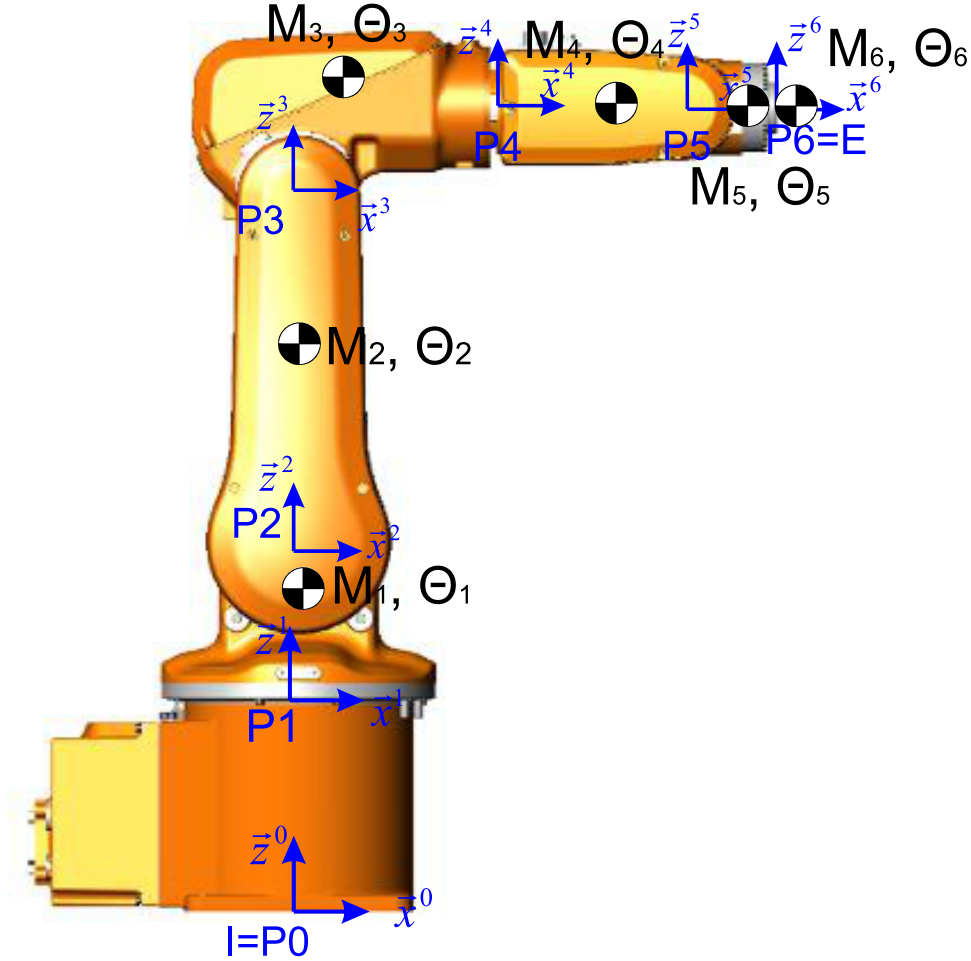

Figure 2: ABB IRB 120 with coordinate systems and joints

2 Model-based control

In this section you will write three controllers which use of the dynamic model of the arm to perform motion and force tracking tasks. The template files can be found in the `problems/` directory. Each controller comes with its own Simulink model, which is stored under `problems/simulink_models/`. To test each of your controllers, open the corresponding model and start the simulation.

2.1 Joint space control

Exercise 2.1

In this exercise you will implement a controller which compensates for the gravitational terms. Additionally, the controller should track a desired joint-space configuration and provide damping which is proportional to the measured joint velocities. Use the provided Simulink block scheme `abb_pd_g.mdl` to test your controller. What behavior would you expect for various initial conditions?

```matlab
function [ tau ] = control_pd_g( q_des, q, q_dot )
% CONTROL_PD_G Joint space PD controller with gravity compensation.
%
% q_des ——> a vector R^n of desired joint angles.
% q ——> a vector R^n of measured joint angles.
% q_dot ——> a vector in R^n of measured joint velocities

% Gains
% Here the controller response is mainly inertia dependent
% so the gains have to be tuned joint-wise
kp = 0; % TODO
kd = 0; % TODO

% The control action has a gravity compensation term, as well as a PD
% feedback action which depends on the current state and the desired
% configuration.
tau = zeros(6,1); % TODO

end
```

Solution 2.1

The control law can be implemented as:

$$\boldsymbol{\tau} = \mathbf{g}(\mathbf{q}) + \mathbf{k}_p(\mathbf{q}_d - \mathbf{q}) - \mathbf{k}_d\dot{\mathbf{q}}, \tag{1}$$

with $\mathbf{g}(\mathbf{q})$ the gravitational terms computed from the dynamics, $\mathbf{q}_d$ the desired joint positions, $\dot{\mathbf{q}}$ the measured joint velocities, and $\mathbf{k}_p$ and $\mathbf{k}_d$ the proportional and derivative gain matrices.

2.2 Inverse dynamics control

Exercise 2.2

In this exercise you will implement a controller which uses an operational-space inverse dynamics algorithm, i.e. a controller which compensates the entire dynamics and tracks a desired motion in the operational-space. Use the provided Simulink model stored in `abb_inv_dyn.mdl`. To simplify the way the desired orientation is defined, the Simulink block provides a way to define a set of Euler Angles XYZ, which will be converted to a rotation matrix in the control law script file.

```matlab
function [ tau ] = control_inv_dyn(I_r_IE_des, eul_IE_des, q, q_dot)
% CONTROL_INV_DYN Operational-space inverse dynamics controller ...
    with a PD
% stabilizing feedback term.
%
% I_r_IE_des ——> a vector in R^3 which describes the desired ...
    position of the
%   end-effector w.r.t. the inertial frame expressed in the ...
    inertial frame.
% eul_IE_des ——> a set of Euler Angles XYZ which describe the desired
%   end-effector orientation w.r.t. the inertial frame.
% q ——> a vector in R^n of measured joint angles
```

```
10  % q_dot --> a vector in R^n of measured joint velocities
11
12  % Set the joint-space control gains.
13  kp = 0; % TODO
14  kd = 0; % TODO
15
16  % Find jacobians, positions and orientation based on the current
17  % measurements.
18  I_J_e = I_Je_fun_solution(q);
19  I_dJ_e = I_dJe_fun_solution(q, q_dot);
20  T_IE = T_IE_fun_solution(q);
21  I_r_Ie = T_IE(1:3, 4);
22  C_IE = T_IE(1:3, 1:3);
23
24  % Define error orientation using the rotational vector ...
        parameterization.
25  C_IE_des = eulAngXyzToRotMat(eul_IE_des);
26  C_err = C_IE_des*C_IE';
27  orientation_error = rotMatToRotVec_solution(C_err);
28
29  % Define the pose error.
30  chi_err = [I_r_IE_des - I_r_Ie;
31             orientation_error];
32
33  % PD law, the orientation feedback is a torque around error ...
        rotation axis
34  % proportional to the error angle.
35  tau = zeros(6, 1); % TODO
36
37  end
```

Solution 2.2

We can define a desired acceleration in operational-space as

$$_I\dot{\mathbf{w}}_d = \mathbf{k}_p \cdot {_I}\boldsymbol{\chi}_{\text{err}} - \mathbf{k}_d \cdot {_I}\mathbf{w}. \tag{2}$$

Recalling that

$$_I\dot{\mathbf{w}} = {_I}\mathbf{J}(\mathbf{q})\ddot{\mathbf{q}} + {_I}\dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}}, \tag{3}$$

which yields

$$\ddot{\mathbf{q}} = {_I}\mathbf{J}(\mathbf{q})^+({_I}\dot{\mathbf{w}} - {_I}\dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}}) \tag{4}$$

we can write the control law as

$$\begin{aligned}
\boldsymbol{\tau} &= \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_d + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) \\
&= \mathbf{M}(\mathbf{q})\mathbf{J}(\mathbf{q})^+({_I}\dot{\mathbf{w}} - {_I}\dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}}) + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})
\end{aligned} \tag{5}$$

with $\mathbf{g}(\mathbf{q})$ the gravitational terms computed from the dynamics, $\mathbf{q}_d$ the desired joint positions, $\dot{\mathbf{q}}$ the measured joint velocities, and $\mathbf{k}_p$ and $\mathbf{k}_d$ the proportional and derivative gain matrices.
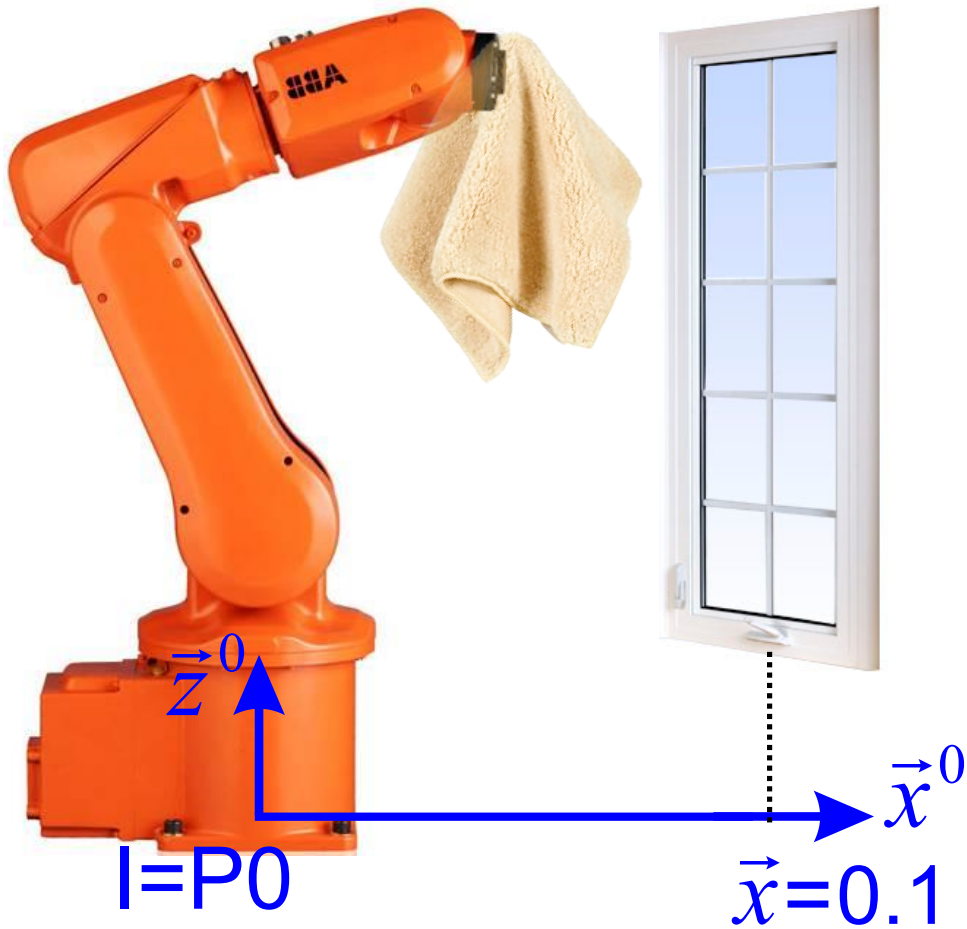
4

Figure 3: Robot arm cleaning a window

2.3 Hybrid force and motion control

Exercise 2.3

We now want to implement a controller which is able to control both motion and force in orthogonal directions by the use of appropriate selection matrices. As shown in Fig. 3, there is a window at $x = 0.1m$. Your task is to write a controller that wipes the window. This controller applies a constant force on the wall in x-axis and follows a trajectory defined on $y - z$ plane. To do this, you should use the equations of motion projected to the operational-space. Use the provided Simulink model abb_op_space_hybrid.mdl, which also implements the reaction force exerted by the window on the end-effector.

```
1  function [ tau ] = control_op_space_hybrid( I_r_IE_des, eul_IE_des, ...
       q, dq, I_F_E_x )
2  % CONTROL_OP_SPACE_HYBRID Operational—space inverse dynamics controller
3  % with a PD stabilizing feedback term and a desired end—effector force.
4  %
5  % I_r_IE_des —> a vector in R^3 which describes the desired ...
       position of the
6  %   end—effector w.r.t. the inertial frame expressed in the ...
```

```matlab
                inertial frame.
 7  % eul_IE_des —-> a set of Euler Angles XYZ which describe the desired
 8  %    end—effector orientation w.r.t. the inertial frame.
 9  % q —-> a vector in R^n of measured joint positions
10  % q_dot —-> a vector in R^n of measured joint velocities
11  % I_F_E_x —-> a scalar value which describes a desired force in the x
12  %    direction
13
14  % Design the control gains
15  kp = 0; % TODO
16  kd = 0; % TODO
17
18  % Desired end—effector force
19  I_F_E = [I_F_E_x, 0.0, 0.0, 0.0, 0.0, 0.0]';
20
21  % Find jacobians, positions and orientation
22  I_Je = I_Je_fun_solution(q);
23  I_dJ_e = I_dJe_fun_solution(q, dq);
24  T_IE = T_IE_fun_solution(q);
25  I_r_IE = T_IE(1:3, 4);
26  C_IE = T_IE(1:3, 1:3);
27
28  % Define error orientation using the rotational vector ...
        parameterization.
29  C_IE_des = eulAngXyzToRotMat(eul_IE_des);
30  C_err = C_IE_des*C_IE';
31  orientation_error = rotMatToRotVec_solution(C_err);
32
33  % Define the pose error.
34  chi_err = [I_r_IE_des — I_r_IE;
35             orientation_error];
36
37  % Project the joint—space dynamics to the operational space
38  % TODO
39  % lambda = ... ;
40  % mu = ... ;
41  % p =  ... ;
42
43  % Define the motion and force selection matrices.
44  % TODO
45  % Sm = ... ;
46  % Sf = ... ;
47
48  % Design a controller which implements the operational—space inverse
49  % dynamics and exerts a desired force.
50  tau = zeros(6,1); % TODO
51
52  end
```

Solution 2.3

According to the task specification, the 6D selection matrices read

$$\mathbf{S}_m = \mathrm{diag}([0,1,1,1,1,1]), \qquad \mathbf{S}_f = \mathrm{diag}([1,0,0,0,0,0]) . \tag{6}$$

Note that their sum equals the identity matrix, i.e., every degree of freedom is either force or position controlled. Now, given the pose error $_I\boldsymbol{\chi}_{\mathrm{err}}$, its derivative $_I\dot{\boldsymbol{\chi}}_{\mathrm{err}}$, and the desired end-effector force of this task $_I\mathbf{f}_{\mathrm{task}}$, we can compute the inverse dynamics control law in operational space as

$$_I\mathbf{f}_{\mathrm{ee,\ des}} = \boldsymbol{\Lambda}\mathbf{S}_m(\mathbf{K}_p\,_I\boldsymbol{\chi}_{\mathrm{err}} - \mathbf{K}_d\,\dot{\boldsymbol{\chi}}_{\mathrm{err}}) + \mathbf{S}_f\,_I\mathbf{f}_{\mathrm{task}} + \boldsymbol{\mu} + \mathbf{p} . \tag{7}$$

This is the virtual force that should act on the robot's end-effector to track the desired motion and force. We can convert this reference to task space via

$$\boldsymbol{\tau} = {}_I\mathbf{J}_{\mathrm{ee}}^{\top}\,_I\mathbf{f}_{\mathrm{ee,\ des}} . \tag{8}$$