MEC6503 Robotique Industrielle
École Polytechnique de Montréal
April 20th 2006, Montréal QC, Canada

PROJECT 3: INVERSE DYNAMICS WITH THE FANUC ARC MATE M16IB

Marie-Ange Janvier **Stefan Bracher**

ABSTRACT

The inverse dynamics of the Fanuc Arc Mate M16iB robot to lift a 2 kg block at 0.2 m vertically in 10 seconds is investigated. To achieve this goal, the trajectory file containing the twist of all links angles and the corresponding time derivatives has been computed offline using the projected resolved-motion rate algorithm. Then kinematics and dynamics computation was achieved to determine the constraints and external wrenches acting on the robot end-effector with the forces and torques exerted by the actuators. The algorithm developed allows to verify if the motors can handle the torques needed to follow the desired path with the dynamics. The results show all joint motors torques required in order for the robot to move the block vertically at 0.2 m for 10 seconds.

INTRODUCTION

The first part of this work is similar to project two "Deburring with the Fanuc M16iB" [1], [2], where we developed the projected resolved-motion rate algorithm to calculate the inverse kinematics (IK) in order to deburr the four corners of a spherical block. In this project, we replace the tool at the end-effector (EE) with a handle. The robot task as well differs, as the robot lifts the square block upward 0.2 m in the same direction for 10 seconds. We still compute the trajectory with the projected resolved-motion rate algorithm to find the twists of all links.

In the second part the inverse dynamics with the purpose to allow real-time dynamic control of the manipulator is introduced. Now we include the twists time derivative of all links and the corresponding mass centers position that are essential to the inverse dynamics kinematics computations. The inverse dynamics also consist of dynamics computations with Newton-Euler algorithm. This algorithm finds the force and torque required by joint motors to complete the task.

In this paper, we first present the path development for that task, the twist of all links and their time derivative and the recursive inverse dynamics to find the joint motors torques input. The results are then presented and discussed.

PATH DEVELOPMENT

It is necessary to develop a path in order to compute the trajectory file for the robot simulator. The trajectory file is the only input to the robot simulator and contains the joint angles that correspond to the points on the path for the task.

The task to lift a block 0.2 m upwards requires the x and y coordinates of the points *pf* on the path to remain constant. Therefore z is the only coordinate moving upward at discrete steps. The step size is varied in order that the step is biggest in the middle of the path and almost zero at the beginning and the end, thus producing smooth velocity changes. Given the orientation is constant when lifting the block, the correlating orientation matrix *Qf* is kept constant during the path. Hence the path defines all *Qf* matrix as:

$$Qf = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix}, \tag{1}$$

with the initial point $pf=[-0.2\ 0.6\ 0.38]^T$ in meters.

THE TWIST TRAJECTORY

The twist trajectory involves finding the twist *t* of all links and their time derivative for all the Cartesian points in the path developed. In the DH parameters, the tool transformation matrix is replaced with the identity orientation matrix and zero translations. This replacement makes the block look as an imaginary 7th robot link, which will help us to include its mass in the dynamic computations later on. The initial twist *t* (or referred to as joint angles) of the manipulator is then found with the DK developed in [3]. Thereafter, all subsequent joint angles θ corresponding to each Cartesian point *pf* in the path are computed with the projected resolved-motion rate algorithm in [2]. The joint angle velocities $\dot{\theta}$ and acceleration $\ddot{\theta}$ then follows by computing the time derivatives of each joint angle θ.

Once all time derivatives of the twist trajectory are known, the recursive dynamics can then be achieved. The mass centers ρ and inertia I of each link i however must be known and defined in the next frame coordinate system F_{i+1}, which is fixed to the body of link i. These parameters are all available in [4].

RECURSIVE INVERSE DYNAMICS

Recursive inverse dynamics algorithm relies on two procedures: the kinematics computations and the dynamics computations. A brief description on both methods is provided below.

Kinematics computations

The kinematics computations make use of an outward recursion algorithm with the Denavit-Hartenberg (DH) notation [5]. The position, velocity and acceleration ($c_i, \dot{c}_i, \ddot{c}_i$) vectors of each link are calculated as well as the angular speed and acceleration vectors ($\omega_i, \dot{\omega}_i$) for all points in a path. Further details on the algorithm are described below.

The outward recursion algorithm is implemented as follows for each link:

1. The inputs to the function are the joint angles with their corresponding time derivatives θ_i, $\dot{\theta}_i$ and $\ddot{\theta}_i$.
2. The 4 x 4 transformation matrix T_i containing the 3 x 3 orientation matrix Q_i and the 3 x 1 link length vector a_i computed with the DH parameters provided in [4] for the current coordinate system F_i.
3. The vector along the z-axis for all revolute joints in F_i is always :
$$e_i = k \ , \tag{2}$$
where k=$[0\ 0\ 1]^T$. We introduce this vector so we can later on build a three dimensional vector out of the one dimensional thetas.

4. The angular velocity ω_i is computed in the next frame coordinate system F_{i+1} as indicated below :
$$\omega_i = Q_i^T \cdot (\omega_{i-1} + \dot{\theta}_i \cdot e_i) \ , \tag{3}$$
with $\omega_0 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$.
ω_{i-1} as well as $\dot{\theta}_i \cdot e_i$ are always expressed in the coordinate system orientation *i*, thus they have to be premultiplied with Q_i^T to bring them in the *i+1* orientation.

5. The angular acceleration $\dot{\omega}_i$ is also computed in the next frame coordinate system F_{i+1} as follows:
$$\dot{\omega}_i = Q_i^T \cdot (\dot{\omega}_{i-1} + \omega_{i-1} \times \dot{\theta}_i \cdot e_i + \ddot{\theta}_i \cdot e_i) \ , \tag{4}$$
with $\dot{\omega}_0 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$.

6. The difference δ_i between link length vector a_i and the mass centers described by the vector ρ_i is constant in F_i but it is transformed in the next frame coordinate system F_{i+1} with Q_i^T as described by:
$$\delta_i = Q_i^T \cdot a_i - \rho_i \ . \tag{5}$$
Where $\delta_0 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$. ρ_i is already defined in F_{i+1} .
7. The position of the origin c_i of the next frame coordinate system F_{i+1} is computed as follows:
$$c_i = Q_i^T \cdot (c_{i-1} + \delta_{i-1}) + \rho_i \ . \tag{6}$$
Where $c_0 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$.
8. The velocity vector $\dot{c}_i$ of the origin and its acceleration vector $\ddot{c}_i$ are also calculated in the next frame coordinate system F_{i+1} with :
$$u_{i-1} = \omega_{i-1} \times \delta_{i-1} \tag{7}$$
$$v_i = \omega_i \times \rho_i \tag{8}$$
giving
$$\dot{c}_i = Q_i^T \cdot (\dot{c}_{i-1} + u_{i-1}) + v_i \tag{9}$$
and
$$\ddot{c}_i = Q_i^T \cdot (\ddot{c}_{i-1} + \dot{\omega}_{i-1} \times \delta_{i-1} + \omega_{i-1} \times u_{i-1}) + \dot{\omega}_i \times \rho_i + \omega_i \times v_i \tag{10}$$
with $\dot{c}_0 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T, u_0 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$ and

$\ddot{c}_0 = [0 \quad 0 \quad g]^T$ to introduce the gravity force *g=-9.81 m/s²* in the manipulator.

9. This process is repeated for all links n of the manipulator for all the points in the path.

The trajectory of the end effector origin and its first and second order derivative ($c_7, \dot{c}_7, \ddot{c}_7$) is provided by the outward recursion algorithm. The procedure can easily be verified, as the second order derivative of the effector origin has to have an "acceleration" of one g in the vertical direction.

The parameters calculated in this algorithm permits dynamic computations to find the joint motors torques necessary with the inward recursion algorithm.

Dynamics computations

The dynamic computation involves the computation of the torques to apply on joint motors for the task at hand. The torques required by the motors is calculated with the Newton-Euler inward recursion algorithm explained in [5]. The inputs to this algorithm are the second order derivatives, the mass centers of the robot members and the block, the angular velocity and acceleration. All these parameters change are evaluated for each joint angle for every point along the path. .

The inward recursion algorithm implemented makes use of: the linear momentum and the conservation of angular momentum principle.

First, the linear forces F applied to each joint are calculated for each point along the path using the principle of linear momentum:

Principle of linear momentum: $m \cdot \ddot{c} = \sum F$

(11)

Giving $F_{joint\, i} = m_i \cdot \ddot{c}_i + F_{joint\, i+1}$

(12)

where *m* defines the mass of the object.

In this way, the inward recursion algorithm for all the joint forces is calculated by beginning with the force on the end effector *F₇*.

As said before, the block is treated like an additional robot member; hence its mass is easily included in the dynamic computations. However, attention still has to be put on the coordinate systems we are working in again. In order to calculate the joint forces in their own coordinate system, the transformation has to be completed the following way:

$$F_i = m_i \cdot Q_i \cdot \ddot{c}_i + Q_i \cdot F_{i+1}\,.$$

(13)

For the joint momentums σ, the "principle of conservation of angular momentum" has to be applied.

Principle of conservation of angular momentum:

$$\dot{L} = \sum M + \sum r \times F$$

(14)

With $\dot{L} = m \cdot r_{OC} \times \ddot{c} + I \cdot \dot{\omega} + \omega \times I \cdot \omega$

M=Momentums (including σ)
rxF=Momentums due to external forces
I=Rotational Inertia

(15)

Thus to find the joint momentums σ_i for each link *i*, the "conservation principle of angular momentum" equation is adjusted as follows:

$$\sigma_i = Q_i(\dot{h}_i + m \cdot \rho_i \times \ddot{c} + Q_i^T \cdot a_i \times F_{i+1} + \sigma_{i+1})$$

(16)

With

$$\dot{h}_i = I_i \cdot \dot{\omega}_i + \omega_i \times I_i \cdot \omega_i$$

(17)

It is important to note that all joint momentums are expressed in the system F_{i+1}. As the coordinate systems have always been chosen so that the z-axis of the following coordinate system F_{i+1} is in the direction of the rotational axis of joint *i*, the motor torques τ is thus equal to the third component of the corresponding σ. Also here the σ_i have to be computed by inward recursion for each point along the path.

The resulting implementation of the inverse dynamics algorithm along with the complete code is available in ANNEX A.

RESULTS

Path developpement

The produced trajectory for the block lifting task is simulated using [6]. The animation shows the manipulator lifting a block of 2kg for a 10 seconds program. Fig 1 illustrates the Cartesian path developed. It can be easily seen that there are no abrupt position changes in the path, so no velocity and acceleration peaks should occur.

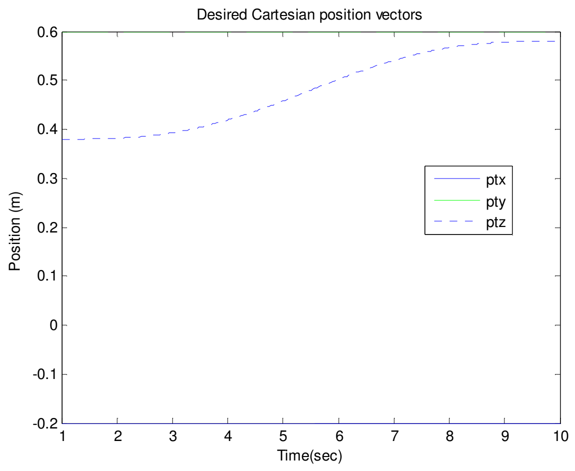

Figure 1: The developed path to lift the block upward

The following figures show some poses for the manipulator along the path designed.

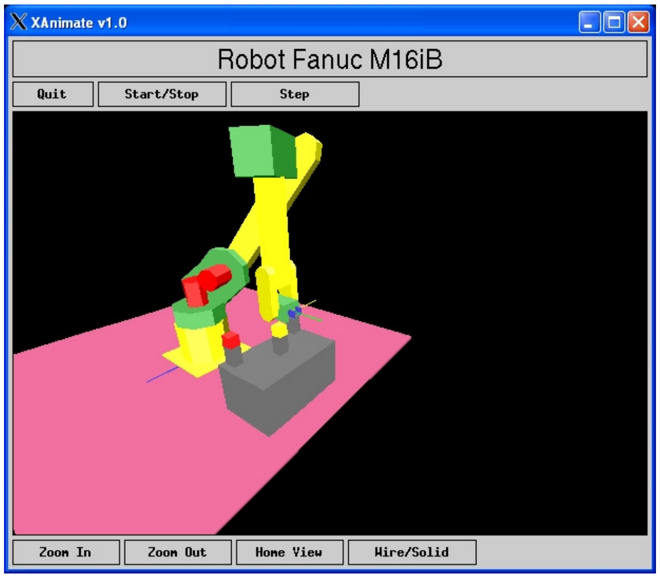

Figure 2: Initial pose of the manipulator

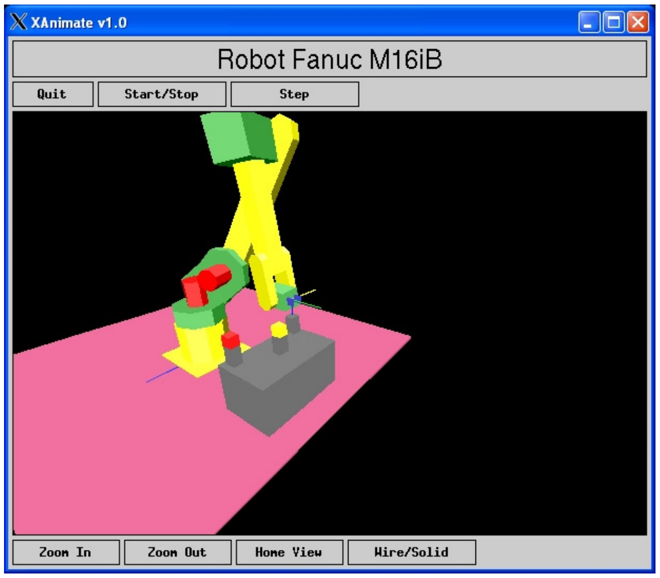

Figure 3: Intermediate pose of the manipulator

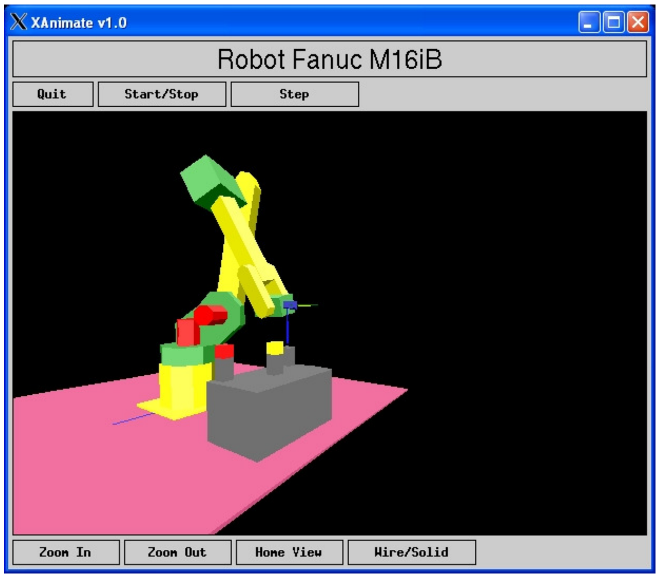

Figure 4: Final pose of the manipulator

Trajectory Twist

The resulting path joint angles θ along with its time derivative $\dot{\theta}$ and $\ddot{\theta}$ are illustrated respectively in Fig.5, Fig.6 and Fig. 7.

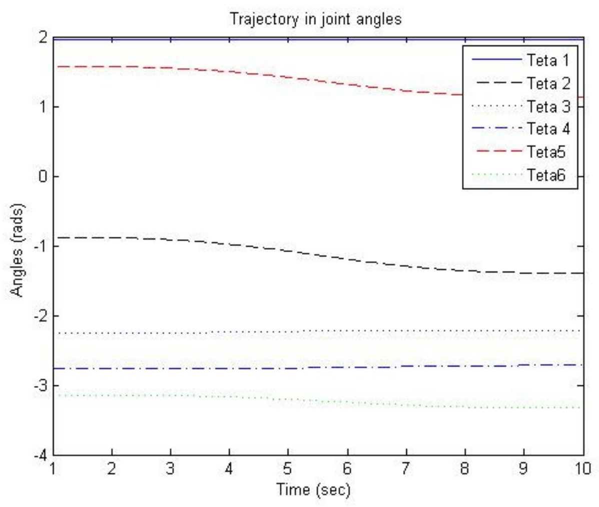

Figure 5: Joint angles trajectories.

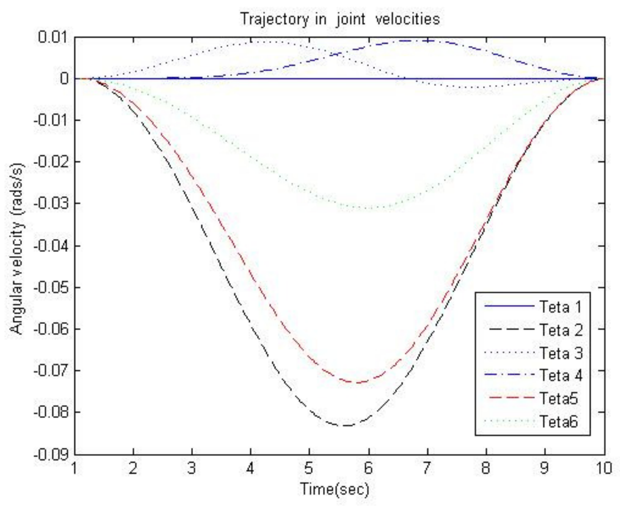

Figure 6: Joint angles velocities

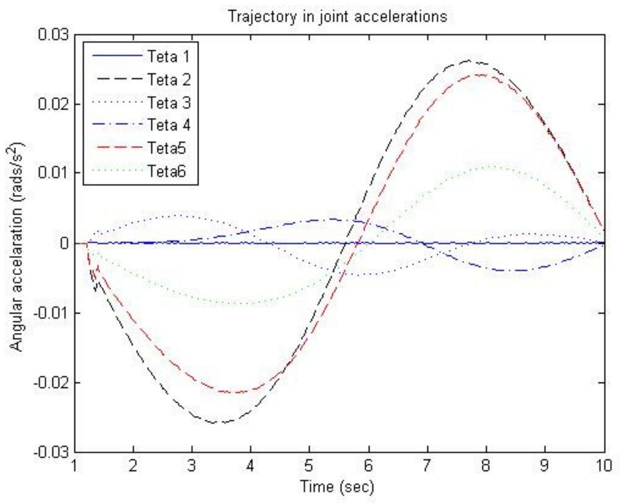

Figure 7: Joint angles acceleration

Trajectory of the end effector origin F_7

Fig. 8-11 show position, velocity and acceleration of the end effector origin. It is to note that the values are given in the referential of the block and not in the robot base. This means x_{block} is pointing in the negative z axis while z_{block} corresponds to the y axis and y_{block} is pointing in the positive direction of the x axis. The entire axes are in reference to the robot base.

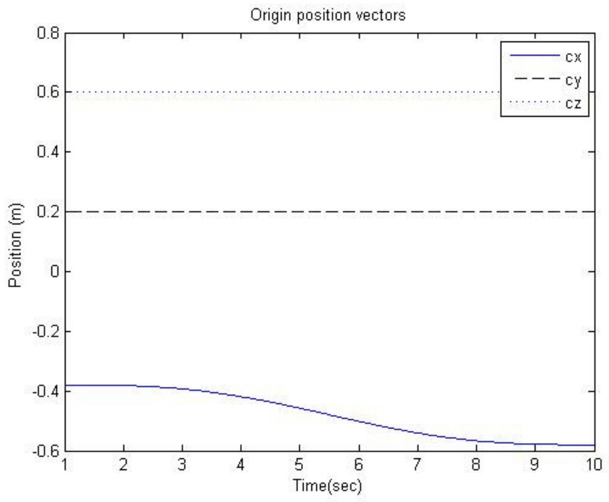

Figure 8: End effector origin position

It is evident in Fig. 8 that the end effector origin moves from x=-0.38 two meters up to x= -0.58 (which corresponds to a movement from z=0.38 to z=0.58 in the robot base system).

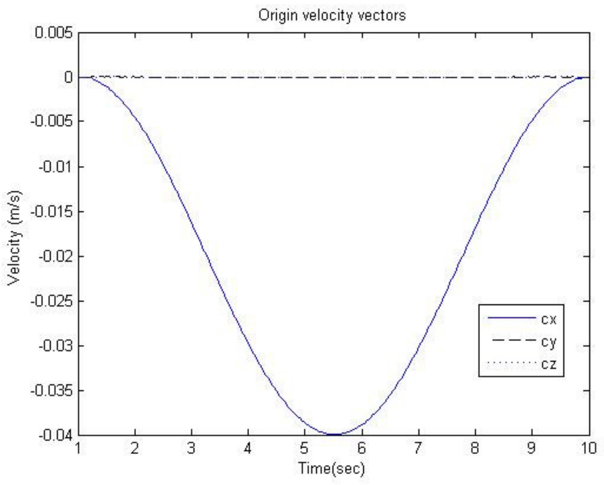

Figure 9: End effector origin velocity

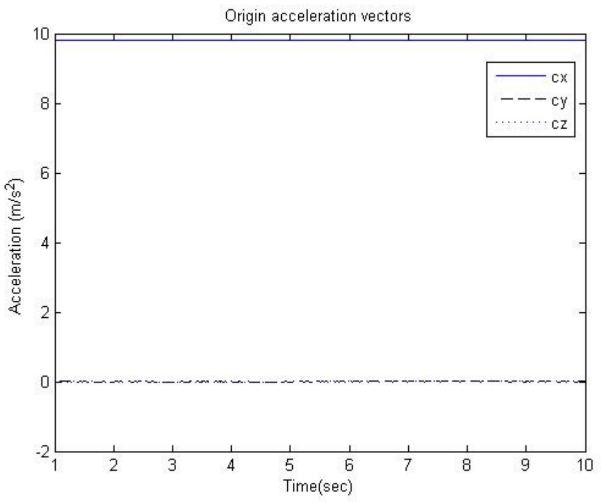

Figure 10: End effector acceleration

It is important to note in Fig. 10 that the force of gravity appears in positive x-direction in the block coordinate system

To provide an overview of the path we include here as well a 3D view in Fig. 11.

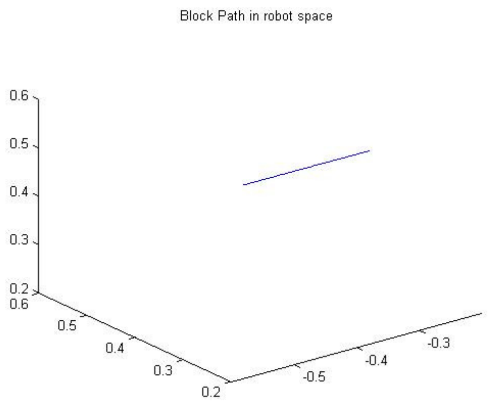

Figure 11: 3D path

Motor torques (Articular couples)

Finally the required motor torques calculated are presented in Fig. 12

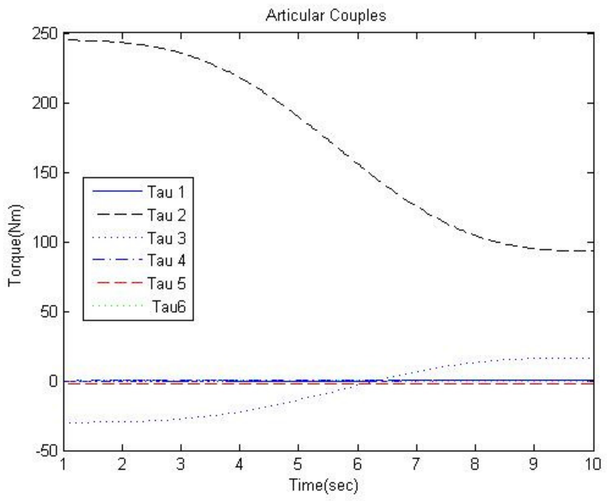

Figure 12: Required motor torques (articular couples)

DISCUSSION

The dynamics computations in lifting a block with a manipulator presented many challenges. Some challenges faced during the development process and the results are discussed.

The calculation of the trajectory joint angles for the movement along the z-axis was supposed to be simple, as we would just reuse the already proven reach-function of the previous project [3]. However some problems occured as the reach-function would always overshoot and never converge. Hence modifications were necessary in the code parameters. The adjustments were made in particular to decrease the damp factor and increase the number of maximal iterations for each point. Afterwards all points converged with the reach function.

To calculate the first and second order derivatives of the joint angles we simply took the differences divided by time. This resulted in some peaks in the derivations due to the very small numbers we were working with. Thus we applied a filter to these derivations, smoothing the calculated results. In the real system, this "low pass filter" would be introduced anyway by the inertias of the system. Acceleration peaks due to mechanical shocks was successfully avoided with the careful step function chosen for path increments.

The most exercised motors were found in the motors in joint 2 and 3. This was expected from the simulation of the path sampled in Fig. 2-4. The joint 1, 4 and 6 have torques close to zero, as due to the geometrical location of these joints they are only facing a torques due to inertia when moving, It is apparent however that the motor 5 always has a torque load around 2 Nm. This is verified by a simple calculation of a torque exercised by the block mass at distance in addition to the member mass multiplied by the distance to the center of gravity:

$$m_6 \cdot \rho_6 \ \cdot 9.81 + m_{block} \cdot 9.81 \cdot a_6 = 0.22 Nm$$

(we neglect in this calculation that joint 6 also rotates, thus giving smaller actual values for torque five) This proves that the torque value at joint 5 is acceptable.

CONCLUSION

In addition to the project one [3] and two [2] already discussed algorithms to calculate joint angles for a desired path, the new algorithms presented here showed how to successfully calculate the required motor torques for a specific task.

With the required motor torques obtained, it could be verified to the motor spec sheets if such a movement is possible with the motors mounted on the manipulator. Or the other way around, knowing the desired task, adequate motors can be chosen.

Finally, this preparation avoids problems with robots not being able to fulfill their task and is thus very important in the modelling of a robot.

REFERENCES

[1] FANUC Robotics America Inc. www.fanucrobotics.com
[2] Bracher, S., Janvier, M.-A., 'Deburring with the a Fanuc M16iB', *École Polytechnique de Montréal*, MEC6503 , March. 16th 2006.
[3] Bracher, S., Janvier, M.-A., 'Postures for a Fanuc M16iB manipulator', *École Polytechnique de Montréal*, MEC6503,10 pages, Feb. 16th 2006.
[4] Baron, L., Task Project3 and class notes, École Polytechnique de Montréal, MEC6503.
[5] Angeles, J. , 1997, Fundamentals of Robotic Mechanical Systems : Theory, Methods, and Algorithms, Springer-Verlag, New-York, second edition
[6] Baron, L., Simulation Software of FANUC robot, École Polytechnique de Montréal

ANNEX A

MATLAB CODE

Tp3.m

```
%TP3 MAIN FILE
%
%Author: Marie-Ange Janvier and Stefan Bracher
%Date:   April 13th, 2006
%


%---------------simulation parameters------------------
dt=0.04;       %robot steptime
t_travel=10;    % Travel time
s_travel=0.2;    % distance to travel
%------------------------------------------------------%


%-----------------desired path and orientation------------------------%
disp('Setting desired Cartesian path and orientation ');
points=t_travel/dt+1;    %number of intermediate steps
ts=[0:dt:t_travel];      %Discretisized time
s=(ts/t_travel)-sin(2*pi*ts/t_travel)/(2*pi); %Discretisized space

pf_z=0.38+0.2*s;            %z coordinates of all points
pf_x=-0.2*ones(1, points);  %x coordinates of all points
pf_y=0.6*ones(1, points);  %y coordinates of all points

PF=[pf_x; pf_y; pf_z];      %Batch of all points


QF=[0 -1 0; 0 0 1; -1 0 0]; %Orientation constant fot all point


%----------------------------------------------------------------------%


% Creating void matrices to fill data in
trj=[];

%---------------Inverse kinematics fot starting point--------------------%

t=[1.951302704 -0.901889446 -2.209582814 -2.760929807 1.598761828 -3.130403303];

% %----------------------- Calculating the articular trajectory--------------------


disp('Calculating joint path');

% joint path displacement
thetas=articulaire(t, dt, PF, QF, points);

thetadots=[zeros(1,6);diff(thetas)]*(1/dt);
a = 1;
b = [1/5 1/5 1/5 1/5];
%Using a low pass filter to reduce peaks due the calculation with very
ythetadots = filter(b,a,thetadots);           %small numbers over 5
thetadotdots=[zeros(1,6);diff(thetadots)]*(1/dt);
%Using a low pass filter to reduce peaks due the calculation with very
ythetadotdots=filter(b,a,thetadotdots);

x=linspace(1,t_travel, points);
 figure
 plot(x', thetas(:,1)); hold on;
 plot(x', thetas(:,2),'k-');
 plot(x', thetas(:,3),':');
 plot(x', thetas(:,4),'-.');
 plot(x', thetas(:,5),'r-');
 plot(x', thetas(:,6),'g:'); hold off;
 legend(,Teta 1','Teta 2','Teta 3','Teta 4', ,Teta5', ,Teta6' );
 Title('Trajectory in joint angles'); xlabel('Time (sec) ');  ylabel('Angles (rads) ');
 figure
 plot(x', ythetadots(:,1)); hold on;
 plot(x', ythetadots(:,2),'k-');
 plot(x', ythetadots(:,3),':');
 plot(x', ythetadots(:,4),'-.');
 plot(x', ythetadots(:,5),'r-');
 plot(x', ythetadots(:,6),'g:'); hold off;
 legend(,Teta 1','Teta 2','Teta 3','Teta 4', ,Teta5', ,Teta6' );
 Title(' Trajectory in  joint  velocities');xlabel('Time(sec) ');  ylabel('Angular velocity (rads/s) ');

 figure
 plot(x', ythetadotdots(:,1)); hold on;
 plot(x', ythetadotdots(:,2),'k-');
 plot(x', ythetadotdots(:,3),':');
 plot(x', ythetadotdots(:,4),'-.');
 plot(x', ythetadotdots(:,5),'r-');
 plot(x', ythetadotdots(:,6),'g:'); hold off;
 legend(,Teta 1','Teta 2','Teta 3','Teta 4', ,Teta5', ,Teta6' );
 Title('Trajectory in joint accelerations');xlabel('Time (sec)');  ylabel('Angular □imulator□on (rads/s^2) ');



%-------------------Calculating the articular couples-------------
disp('Calculating the articuar couples now');
for i=1:points

[w, wdot, c, cdot, cdotdot]=outward_recursion2(thetas(i, :), thetadots(i, :), thetadotdots(i, :));

tau=inward_recursion2(cdotdot,thetas(i, :) w, wdot);

%------STORING THE DATA SO IT CAN BE SEEN IN WORKSPACE------------%
tau_NE(i, :)=tau;

w1in2(i, :) =w(1:3, 1)';
w2in3(i, :) =w(1:3, 2)';
w3in4(i, :) =w(1:3, 3)';
w4in5(i, :) =w(1:3, 4)';
w5in6(i, :) =w(1:3, 5)';
w6in7(i, :)=w(1:3, 6)';
w7in8(i, :)=w(1:3, 7)';

wdot1in2(i, :)=wdot(1:3, 1)';
wdot2in3(i, :)=wdot(1:3, 2)';
wdot3in4(i, :)=wdot(1:3, 3)';
wdot4in5(i, :)=wdot(1:3, 4)';
wdot5in6(i, :)=wdot(1:3, 5)';
wdot6in7(i, :)=wdot(1:3, 6)';
wdot7in8(i, :)=wdot(1:3, 7)';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%VERIFICATION: wdots start at zero and end at zero%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

c1in2(i, :)=c(1:3, 1)';
c2in3(i, :)=c(1:3, 2)';
c3in4(i, :)=c(1:3, 3)';
c4in5(i, :)=c(1:3, 4)';
c5in6(i, :)=c(1:3, 5)';
c6in7(i, :)=c(1:3, 6)';
c7in8(i, :)=c(1:3, 7)';
%NOTE: c7in8 describse the origin 7 in ORIENTATION 8 (AND NOT COORDINATE
%SYSTEM 8)
%A look at the □imulator tells us that x8=-z0, y8=-x0 and z8=y0
%(This could also be optaine with changing the orientation using Q1Q2..ect)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%VERIFICATION:  c7 travels 0.2m in -x8 direction, what is equal
%               to moving 200mm up ->OK
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


cdot1in2(i, :)=cdot(1:3, 1)';
cdot2in3(i, :)=cdot(1:3, 2)';
cdot3in4(i, :)=cdot(1:3, 3)';
cdot4in5(i, :)=cdot(1:3, 4)';
cdot5in6(i, :)=cdot(1:3, 5)';
```

```
cdot6in7(i, :)=cdot(1:3, 6)';
cdot7in8(i, :)=cdot(1:3, 7)';

cdotdot1in2(i, :)=cdotdot(1:3, 1)';
cdotdot2in3(i, :)=cdotdot(1:3, 2)';
cdotdot3in4(i, :)=cdotdot(1:3, 3)';
cdotdot4in5(i, :)=cdotdot(1:3, 4)';
cdotdot5in6(i, :)=cdotdot(1:3, 5)';
cdotdot6in7(i, :)=cdotdot(1:3, 6)';
cdotdot7in8(i, :)=cdotdot(1:3, 7)';
%---------------END OF DATA STORING------------%

end

figure
 plot(x', tau_NE( :,1)); hold on;
 plot(x', tau_NE( :,2),'k-');
 plot(x', tau_NE( :,3),' :');
 plot(x', tau_NE( :,4),'-.');
 plot(x', tau_NE( :,5),'r-');
 plot(x', tau_NE(:,6),'g:'); hold off;
 legend('Tau 1','Tau 2','Tau 3','Tau 4', 'Tau 5', ' Tau6' );
 Title('Articular Couples');xlabel('Time(sec) ');  ylabel('Torque(Nm) ');


 c7=c7in8;
 c7dots=cdot7in8;
 c7dotdots=cdotdot7in8;

 x=linspace(1,t_travel, points);
 figure
 plot(x',c7(:,1)); hold on;
 plot(x',c7(:,2),'k-');
 plot(x', c7(:,3),':');
 hold off;
 legend('cx','cy','cz' );xlabel('Time(sec)');  ylabel('Position (m) ');
 Title('Origin position vectors');

 figure
 plot3(c7(:,1),c7(:,2),c7(:,3));
AXIS([-0.6 -0.2 0.2 0.6 0.2 0.6])

 Title('Block Path in robot space ');


 figure
 plot(x', c7dots(:,1)); hold on;
 plot(x', c7dots(:,2),'k-');
 plot(x', c7dots(:,3),':');
 ; hold off;
 legend('cx','cy','cz');xlabel('Time(sec)');  ylabel('Velocity (m/s) ');
Title('Origin velocity vectors');

 figure
 plot(x', c7dotdots(:,1)); hold on;
 plot(x',c7dotdots(:,2),'k-');
 plot(x', c7dotdots(:,3),':');
  hold off;
 legend('cx','cy','cz');xlabel('Time(sec)');  ylabel('Acceleration (m/s^2) ');
 Title('Origin acceleration vectors');
```

Outward_recursion2.m

```
function  [w, wdot, c, cdot, cdotdot]=outward_recursion2(t, tdot, tdotdot)
% c, cdot, cdotdot, wdot,
%------------------obtaining system information-------------------
[a, b, alpha, alpha_min, alpha_max,rho]=david_hartenberg;   %Loading the D-H description
T_tool=tool;                                               %Loading the tool transformation matrix

T1=transformation_matrix(alpha(1), t(1), a(1), b(1));   %Transformation matrix of the first member
T2=transformation_matrix(alpha(2), t(2), a(2), b(2));   %Transformation matrix of the 2nd member
T3=transformation_matrix(alpha(3), t(3), a(3), b(3));   %Transformation matrix of the 3d member
T4=transformation_matrix(alpha(4), t(4), a(4), b(4));   %Transformation matrix of the 4th member
T5=transformation_matrix(alpha(5), t(5), a(5), b(5));   %Transformation matrix of the 5th member
T6=transformation_matrix(alpha(6), t(6), a(6), b(6));   %Transformation matrix of the 6th member

Q1=T1(1:3, 1:3); %Extracting the orientation part
Q2=T2(1:3, 1:3); %Extracting the orientation part
Q3=T3(1:3, 1:3); %Extracting the orientation part
Q4=T4(1:3, 1:3); %Extracting the orientation part
Q5=T5(1:3, 1:3); %Extracting the orientation part
Q6=T6(1:3, 1:3); %Extracting the orientation part
Q7=[1 0 0; 0 1 0; 0 0 1];       % Fake Q7 for the block as a "robot member

e=[0 0 1]';     %Vector along z-axis which is ALWAYS used as the rotational Axis
                %So it can be used to build a vektor out of the thetas



%---------The omegas----------------------------%
%%%%%%EACH OMEGA IS WRITTEN IN THE NEXT COORDONATE SYSTEM%%%%%%%%%%%%%%%%%%
%General formula for revolute joints: omega=omegaprevious + theta*[0 0 1]'
w(:, 1)=Q1'*(tdot(1)*e);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Here we can do a little test: Q1*w(:, 1) should be equal to Theta1*e   %
% disp('test w1:')                                                      %
% Q1*w(:, 1)                                                            %
% t(1)                                %OK                               %
%(Note to ourselves: don't forget to put that in the report!!)          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

w(:, 2)=Q2'*(w(1:3,1)+tdot(2)*e);
w(:, 3)=Q3'*(w(1:3,2)+tdot(3)*e);
w(:, 4)=Q4'*(w(1:3,3)+tdot(4)*e);
w(:, 5)=Q5'*(w(1:3,4)+tdot(5)*e);
w(:, 6)=Q6'*(w(1:3,5)+tdot(6)*e);
w(:, 7)=Q7'*w(1:3,6);

%Test if ok
% endeffector=Q1*Q2*Q3*Q4*Q5*Q6*w(:, 6)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%---------The omega dots----------------------------%
%%%%%%EACH OMEGA IS WRITTEN IN THE NEXT COORDONATE SYSTEM%%%%%%%%%%%%%%%%%%
%General formula for revolute joints: omegadot=omegadotprevious + omegaprev_x_thetadot+ thetadotdot
wdot(:, 1)=Q1'*(tdotdot(1)*e);
wdot(:, 2)=Q2'*(wdot(1:3,1)+cross(w(:, 1), tdot(2)*e)+tdotdot(2)*e);
wdot(:, 3)=Q3'*(wdot(1:3,2)+cross(w(:, 2), tdot(3)*e)+tdotdot(3)*e);
wdot(:, 4)=Q4'*(wdot(1:3,3)+cross(w(:, 3), tdot(4)*e)+tdotdot(4)*e);
wdot(:, 5)=Q5'*(wdot(1:3,4)+cross(w(:, 4), tdot(5)*e)+tdotdot(5)*e);
wdot(:, 6)=Q6'*(wdot(1:3,5)+cross(w(:, 5), tdot(6)*e)+tdotdot(6)*e);
wdot(:, 7)=Q7'*wdot(1:3,6);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%---------The deltas- A helping variable---------------------------%
delta(:,1)=Q1'*(T1(1:3,4))-rho(:,1);
delta(:,2)=Q2'*(T2(1:3,4))-rho(:,2);
delta(:,3)=Q3'*(T3(1:3,4))-rho(:,3);
delta(:,4)=Q4'*(T4(1:3,4))-rho(:,4);
delta(:,5)=Q5'*(T5(1:3,4))-rho(:,5);
delta(:,6)=Q6'*(T6(1:3,4))-rho(:,6);    %normal robot members
delta(:,7)=0;                         %the artificial additional member "block"

%---------The c's---------------------------%
%What the thing is doing:
%It calculates the position of ci in the ORIENTATION i+1
%(THIS IS NOT THE SYSTEM i+1, JUST IT'S ORIENTATION
%BUT IT DOES NOT MATTER AS WE JUST NEED THE DERIVATION OF C)
%
%(ACUTALLY THE COMPUTATION OF C IS ABSOLUTELY USELESS, WE JUST DO IT BECAUSE
%F7(c) IS ASKED IN THE ASSIGNEMENT)
%
%It starts at the coordinate origin of i-1 exressed in i
%(equals "c(:,i-1)+delta(:,i-1)", see picture Classbook Page 315)
%Adds rho to get to the center of gravity of i
%And finaly changes the orientation from i to i+1
%(Premultiplication with Qi')
%

%THE BOOKS ORIGINAL VERSION%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c(:, 1)=rho(:,1);
c(:, 2)=Q2'*(c(:, 1)+delta(:,1))+ rho(:,2);
c(:, 3)=Q3'*(c(:, 2)+delta(:,2))+ rho(:,3);
c(:, 4)=Q4'*(c(:, 3)+delta(:,3))+ rho(:,4);
c(:, 5)=Q5'*(c(:, 4)+delta(:,4))+ rho(:,5);
c(:, 6)=Q6'*(c(:, 5)+delta(:,5))+ rho(:,6);
c(:, 7)=Q7'*(c(:, 6)+delta(:,6));           % Tge block
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%---------u, v, cdot and cdotdot with Qi'*rho(i)---------------------------%
cdot_zero=[0 0 0]';
cdotdot_zero=[0 0 -9.81]';   %So we introduce g-force, SIGN might be inverted, Q1 might needed
u_zero=0;
w(:, 1);
v(:, 1)=cross(w(:, 1), rho(:,1));  %PROBABLY Q1'rho... needed as before to put vi in i+1
cdot(:, 1)=v(:,1);
cdotdot(:, 1)=Q1'*(cdotdot_zero)+cross(wdot(:,1), rho(:, 1))+cross(w(:,1), v(:, 1)); %PROBABLY Q1'rho...

%for membrure 2
u(:, 1)=cross(w(:, 1), delta(:, 1));
v(:, 2)=cross(w(:, 2), rho(:,2));  %PROBABLY Q2'rho... needed as before to put vi in i+1
cdot(:, 2)=Q2'*(cdot(:, 1)+u(:, 1))+v(:, 2);
cdotdot(:, 2)=Q2'*(cdotdot(:, 1)+cross(wdot(:,1), delta(:, 1))+cross(w(:,1), u(:, 1)))+cross(wdot(:,2), rho(:,
2))+cross(w(:,2), v(:, 2));
%PROBABLY Q2'rho2 needed..

%for membrure 3
u(:, 2)=cross(w(:, 2), delta(:, 2));
v(:, 3)=cross(w(:, 3), rho(:,3));  %PROBABLY Q3'rho... needed as before to put vi in i+1
cdot(:, 3)=Q3'*(cdot(:, 2)+u(:, 2))+v(:, 3);
cdotdot(:, 3)=Q3'*(cdotdot(:, 2)+cross(wdot(:,2), delta(:, 2))+cross(w(:,2), u(:, 2)))+cross(wdot(:,3), rho(:,
3))+cross(w(:,3), v(:, 3));
%PROBABLY Q3'rho3 needed..

%for membrure 4
u(:, 3)=cross(w(:, 3), delta(:, 3));
v(:, 4)=cross(w(:, 4), rho(:,4));  %PROBABLY Q4'rho... needed as before to put vi in i+1
cdot(:, 4)=Q4'*(cdot(:, 3)+u(:, 3))+v(:, 4);
cdotdot(:, 4)=Q4'*(cdotdot(:, 3)+cross(wdot(:,3), delta(:, 3))+cross(w(:,3), u(:, 3)))+cross(wdot(:,4), rho(:,
4))+cross(w(:,4), v(:, 4));
%PROBABLY Q4'rho4 needed..

%for membrure 5
u(:, 4)=cross(w(:, 4), delta(:, 4));
v(:, 5)=cross(w(:, 5), rho(:,5));  %PROBABLY Q5'rho... needed as before to put vi in i+1
cdot(:, 5)=Q5'*(cdot(:, 4)+u(:, 4))+v(:, 5);
cdotdot(:, 5)=Q5'*(cdotdot(:, 4)+cross(wdot(:,4), delta(:, 4))+cross(w(:,4), u(:, 4)))+cross(wdot(:,5), rho(:,
5))+cross(w(:,5), v(:, 5));
%PROBABLY Q5'rho5 needed..

%for membrure 6
u(:, 5)=cross(w(:, 5), delta(:, 5));
v(:, 6)=cross(w(:, 6), rho(:,6));  %PROBABLY Q6'rho... needed as before to put vi in i+1
cdot(:, 6)=Q6'*(cdot(:, 5)+u(:, 5))+v(:, 6);
cdotdot(:, 6)=Q6'*(cdotdot(:, 5)+cross(wdot(:,5), delta(:, 5))+cross(w(:,5), u(:, 5)))+cross(wdot(:,6), rho(:,
6))+cross(w(:,6), v(:, 6));
```

```
%PROBABLY Q6'rho6 needed..

%for fake membrure 7, the block
u(:, 6)=cross(w(:, 6), delta(:, 6));
v(:, 7)=[0 0 0]';
cdot(:, 7)=Q7'*(cdot(:, 6)+u(:, 6))+v(:, 7);
cdotdot(:, 7)=Q7'*(cdotdot(:, 6)+cross(wdot(:,6), delta(:, 6))+cross(w(:,6), u(:, 6)))++cross(w(:,7), v(:, 7));
%




end
```

inward_recursion

```
function tau=inward_recursion(cdotdot,t, w, wdot)
%%%%%%%%%%FIRST ACUIRING SOME GENERAL SYSTEM INFORMATION%%

%total number of links
n=7;  %6 for the robot, 1 to include the block as a "robot member"

% Dynamic parameters
m=[39 27 25 15 2.5 0.5 2]; %g=-9.81;

%David hartenmerg
[a, b, alpha, alpha_min, alpha_max, rho]=david_hartenberg;

T1=transformation_matrix(alpha(1), t(1), a(1), b(1));   %Transformation matrix of the first member
T2=transformation_matrix(alpha(2), t(2), a(2), b(2));   %Transformation matrix of the 2nd member
T3=transformation_matrix(alpha(3), t(3), a(3), b(3));   %Transformation matrix of the 3d member
T4=transformation_matrix(alpha(4), t(4), a(4), b(4));   %Transformation matrix of the 4th member
T5=transformation_matrix(alpha(5), t(5), a(5), b(5));   %Transformation matrix of the 5th member
T6=transformation_matrix(alpha(6), t(6), a(6), b(6));   %Transformation matrix of the 6th member

Q1=T1(1:3, 1:3); %Extracting the orientation part
Q2=T2(1:3, 1:3); %Extracting the orientation part
Q3=T3(1:3, 1:3); %Extracting the orientation part
Q4=T4(1:3, 1:3); %Extracting the orientation part
Q5=T5(1:3, 1:3); %Extracting the orientation part
Q6=T6(1:3, 1:3); %Extracting the orientation part
Q7=[1 0 0; 0 1 0; 0 0 1];       % Fake Q7 for the block as a "robot member

I1=inertia(1);
I2=inertia(2);
I3=inertia(3);
I4=inertia(4);
I5=inertia(5);
I6=inertia(6);
I7=inertia(7);

%%%%%%%%%%%%%%%%%SOME THEORY%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%IMPULSSATZ:  m*cdotdot=SUM of all outher forces    %%%
%%%             -> m*cdotdot=Fi-Fi+1                  %%%
%%%             ->Fi=m*cdotdot+Fi+1                   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-------Using Impulssatz we can calculate all forces aplied to each joint--
f7=m(7)*Q7*cdotdot(1:3, 7);  %As cdotdot is in i+1, pre-multiplication with Qi brings it in i, f is in i
f6=m(6)*Q6*cdotdot(1:3, 6)+Q6*f7;  %f6 will be in i=6
f5=m(5)*Q5*cdotdot(1:3, 5)+Q5*f6;
f4=m(4)*Q4*cdotdot(1:3, 4)+Q4*f5;
f3=m(3)*Q3*cdotdot(1:3, 3)+Q3*f4;
f2=m(2)*Q2*cdotdot(1:3, 2)+Q2*f3;
f1=m(1)*Q1*cdotdot(1:3, 1)+Q1*f2;
%All forces are in their own coordonate orientations

%A little test at this place:
%As robot stopped at the end, force f1 should be equal to the sum
%of all masses multiplied with g
%in negative z in the global system ->ok

%%%%%%%%%%%%%%%%%SOME THEORY AGAOM%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%DRALLSATZ:  Ldot=sum(moments)+sum(moments incuced by forces)  %%%
%%%            L=Impulsmoment+Spin=                              %%%
%%%            Impulsmoment=m*rii x cdot                         %%%
%%%            Spin=I*w                    ->h                   %%%
%%%            m.by forces=ai x(-f(i+1))                         %%%
%%%            moments=s(i)-s(i+1)                               %%%
%%%                                                              %%%
%%% thus:   (m*rii x cdot)dot+hdot=ai x(-f(i+1))+s(i)-s(i+1)     %%%
%%% ->      s(i)=hdot+(m*rii x cdot)dot+ai x f(i+1)+s(i+1)       %%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%--------using impulssath to calculate the moments s(i) in joint i-----
hdot7=I7*wdot(1:3,7)+cross(w(1:3, 7), I7*w(1:3, 7));    %As suggested by the book
hdot6=I6*wdot(1:3,6)+cross(w(1:3, 6), I6*w(1:3, 6));
hdot5=I5*wdot(1:3,5)+cross(w(1:3, 5), I5*w(1:3, 5));
hdot4=I4*wdot(1:3,4)+cross(w(1:3, 4), I4*w(1:3, 4));
hdot3=I3*wdot(1:3,3)+cross(w(1:3, 3), I3*w(1:3, 3));
hdot2=I2*wdot(1:3,2)+cross(w(1:3, 2), I2*w(1:3, 2));
hdot1=I1*wdot(1:3,1)+cross(w(1:3, 1), I1*w(1:3, 1));
%hdot in system i+1

s7=[0 0 0]';        % should be zero as spin of a point always is
s6=Q6*(hdot6+cross(m(6)*rho(:, 6), cdotdot(1:3, 6))+cross(Q6'*T6(1:3, 4), f7)+s7);  %in sys 6
%A little test:
%test=Q6'*s6 %=0.21 should give the 3d torque in the end-effector coordonate system
% should give around 0.22 (m6*rho6*g+mblock*g*f) around y axis OK

s5=Q5*(hdot5+cross(m(5)*rho(:, 5), cdotdot(1:3, 5))+cross(Q5'*T5(1:3, 4), f6)+s6); % in sys 5
%test 2
%test2=Q5'*Q6'*s5 %gives x 0.41, y -2.16, z -0.08
%should give around -2kg*g*0.1m (=1.962) - 0.5kg*9.81*0.05m (=0.24) = -2.2
%around y OK (difference due to not perfect perpendicularity of g and z)

s4=Q4*(hdot4+cross(m(4)*rho(:, 4), cdotdot(1:3, 4))+cross(Q4'*T4(1:3, 4), f5)+s5); % in sys 4

s3=Q3*(hdot3+cross(m(3)*rho(:, 3), cdotdot(1:3, 3))+cross(Q3'*T3(1:3, 4), f4)+s4); % in sys 3
s2=Q2*(hdot2+cross(m(2)*rho(:, 2), cdotdot(1:3, 2))+cross(Q2'*T2(1:3, 4), f3)+s3); % in sys 2
s1=Q1*(hdot1+cross(m(1)*rho(:, 1), cdotdot(1:3, 1))+cross(Q1'*T1(1:3, 4), f2)+s2); % in sys 1



tau=[s1(3), s2(3), s3(3), s4(3), s5(3), s6(3)];  %Tau is always the value for z as z is always along the
rotation axis
```

david_hartenberg.m

```
function  [a, b, alpha, alpha_min, alpha_max, rho]=david_hartenberg()

%Defines the David-Hartenberg Parameters of the Robot

a=[0.15,
   0.770,
   0.10,
   0,
   0,
   0];

b=[0.525,
   0,
   0,
   0.740,
   0,
   0.10];
alpha=[-pi/2,
        0,
        pi/2,
        -pi/2,
        pi/2,
        0];
alpha_min=[-2.9671,
           -2.1817,
           -4.0143,
           -3.4907,
           -2.4435,
           -7.8540];
alpha_max=[2.9671,
           2.1817,
           4.0143,
           3.4907,
           2.4435,
           7.8540];

rho=[ 0.07, 0.35, 0.05,     0, 0, 0 ;
     -0.26,    0,    0, -0.35, 0, 0 ;
         0,    0, 0.02,     0, 0, 0.05 ];
```

Transformation_matrix.m

```
function  T=transformation_matrix(alpha, theta, a, b)
%Calculates the tranformation matrix of a robot member
%acording to the formula defined in the Report of TP1 Page 2

T=[cos(theta) -cos(alpha)*sin(theta) sin(alpha)*sin(theta) a*cos(theta);
   sin(theta) cos(alpha)*cos(theta) -sin(alpha)*cos(theta) a*sin(theta);
   0 sin(alpha) cos(alpha) b;
   0 0 0 1
];
```

Tool.m

```
function  T_tool=tool()
%Defines the tool tranformation matrix of the robot

% beta=-25.2*pi/180;
% T_tool=[cos(beta), 0, -sin(beta), -0.0785;...
%              0, 1, 0, 0; ...
%              sin(beta), 0, cos(beta), 0.154;...
%              0, 0, 0, 1];

% T_tool=[1, 0, 0, 0.1;...
%          0, 1, 0, 0; ...
%          0, 0, 1, 0;...
%          0, 0, 0, 1];

T_tool=[1, 0, 0, 0;...
        0, 1, 0, 0; ...
        0, 0, 1, 0;...
        0, 0, 0, 1];
```

Inertia.m

```
function I=inertia(value)

if (value == 1)
I=[3.1 0 0
    0 3.0 0
    0 0 3];
end
if (value == 2)
I=[0.3 0 0
    0 2.9 0
    0 0 3];
end
if (value == 3)
I=[1.1 0 0
    0 1.1 0
    0 0 1.1];
end
if (value == 4)
I=[3.0 0 0
    0 0.3 0
    0 0 3];
end
if (value == 5)
I=[0.8 0 0
    0 0.8 0
    0 0 0.1];
end
if (value==6)
I=[0.2 0 0
    0 0.2 0
    0 0 0.1];
end
if (value==7)
I=[0 0 0
   0 0 0
   0 0 0];
end
end
```

reach.m

```
function [t, label]=reach(pf,Qf, t)
% inputs to evaluate new postion of tool for the manipulator
% ep=erreur positoin, ee=erreur d'orientation n= nombre d`iterations
% a=amortisseur

n=500; a= 0.1; ep=1; ee=10;


[p,Q,J]=mgd(t); %actual position, orientation and Jacobienne
ep=norm(pf-p);
ee=asin(norm(Q*vect(Q'*Qf)));
e=Q(1:3, 3);
P=eye(3)-e*e'; L =e*e';

T=zeros(6,6);
% 0 orientation, 1 positions
 %T(1:3,1:3)=zeros(3); T(4:6,4:6)=L;

% 0 orientation, 2 positions
% T(1:3,1:3)=zeros(3); T(4:6,4:6)=P;

% 0 orientation, 3 positions
% T(1:3,1:3)=zeros(3); T(4:6,4:6)=eye(3);



% 1 orientation, 1 positions
 %T(1:3,1:3)=L; T(4:6,4:6)=L;

% 1 orientation, 2 positions
% T(1:3,1:3)=L; T(4:6,4:6)=P;

% 1 orientation, 3 positions
% T(1:3,1:3)=L; T(4:6,4:6)=eye(3);

% 2 orientation, 1 positions
 %T(1:3,1:3)=P; T(4:6,4:6)=L;

%2 orientation, 2 positions
%T(1:3,1:3)=P; T(4:6,4:6)=P;

% 2 orientation, 3 positions
% T(1:3,1:3)=P; T(4:6,4:6)=eye(3);

% 3 orientation, 1 positions
%T(1:3,1:3)=eye(3); T(4:6,4:6)=L;

%3 orientation, 2 positions
%T(1:3,1:3)=eye(3); T(4:6,4:6)=P;

% 3 orientation, 3 positions
T(1:3,1:3)=eye(3); T(4:6,4:6)=eye(3);


[z, z, alpha, alpha_min, alpha_max,rho]=david_hartenberg;
tmax=alpha_min';
tmin=alpha_max';
tmean=1/2*(tmax + tmin);
W=diag(tmax-tmin);
h=-W*(t-tmean)';

% while(n>0 & (ep>0.0001 | ee> 0.001))
    while(n>0 & (ep>0.00001 | ee> 0.0001))

    dp=a*(pf-p);     %dp

    dq=a*Q*vect(Q'*Qf);
    dx=[dq;dp];
    dt=(pinv(J)*T)*dx + pinv(J)*(eye(6)-T)*J*h;
    % dt=pinv(J)*dx + (eye(6)-pinv(J)*J)*h;
    t=t+dt';
    %Correcting overshoots
    for ti=1:6
     while t(ti)>(2*pi)
         t(ti)=t(ti)-(2*pi);
     end
     while t(ti)<(-2*pi)
         t(ti)=t(ti)+(2*pi);
     end
    end

    [p,Q,J]=mgd(t); %actual position
    ep=norm(pf-p);
    ee=asin(norm(Q*vect(Q'*Qf)));
    n=n-1;

end
label=1;

if n==0
        disp('reach error');
        label=0;
end
```

mgd.m

```
function  [p,Q,J]=mgd(thetas)
%Calculates the direkt kinematics of the Robot

%-----------functions needed---------------------------------------
%david_hartenberg.m     The D-H-description of the robot
%tool.m                 The tool tranformation matrix
%transformation_matrix.m Buids tha transf.matrix of a robot member
%------------------------------------------------------------------



[a, b, alpha, alpha_min, alpha_max,rho]=david_hartenberg;   %Loading the D-H description
T_tool=tool;                                               %Loading the tool transformation matrix

T1=transformation_matrix(alpha(1), thetas(1), a(1), b(1));   %Transformation matrix of the first member
T2=transformation_matrix(alpha(2), thetas(2), a(2), b(2));   %Transformation matrix of the 2nd member
T3=transformation_matrix(alpha(3), thetas(3), a(3), b(3));   %Transformation matrix of the 3d member
T4=transformation_matrix(alpha(4), thetas(4), a(4), b(4));   %Transformation matrix of the 4th member
T5=transformation_matrix(alpha(5), thetas(5), a(5), b(5));   %Transformation matrix of the 5th member
T6=transformation_matrix(alpha(6), thetas(6), a(6), b(6));   %Transformation matrix of the 6th member

Ttot=T1*T2*T3*T4*T5*T6*T_tool;                              %Calculating the overall transformation matrix

p=Ttot(1:3,4);                                              %Extracting the end effector position
Q=Ttot(1:3,1:3);                                            %Extracting the end effector orientation

%UNTIL HERE EVERYTHING IS FINE AND WORKING!!!!!!

% now we will get the Jacobian at this point

%first the orientation matrices are build
 Q11=T1(1:3,1:3);        Q12=Q11*T2(1:3,1:3);
 Q13=Q12*T3(1:3,1:3);    Q14=Q13*T4(1:3,1:3);
 Q15=Q14*T5(1:3,1:3);    Q16=Q15*T6(1:3,1:3);
 %Q17=Q16*T_tool;
%Q17 TO END EFFECTOR NEEDED? WE ARE HERE JUST AT THE END EFFECTOR BASE!!!!!

%The unitary vectors relative to the base are calculated

 k=[0, 0, 1]';  %Definition

 e1=k;
 e2=Q11*k;
 e3=Q12*k;

 e4=Q13*k;
 e5=Q14*k;
 e6=Q15*k;
%  e7=Q16*k;

 %ALSO HERE WE GO JUST TO THE END EFFECTOR BASE!!!!!


 %The radii ri defined on the previous referential are obtained
 %ALSO HERE WE GO JUST TO THE END EFFECTOR BASE!!!!!
%     r7=T_tool(1:3,4);
    r6=T6(1:3,4);
    %r6=T6(1:3,4)+Q16(1:3,1:3)*r7;
    r5=T5(1:3,4)+Q15(1:3,1:3)*r6;
    r4=T4(1:3,4)+Q14(1:3,1:3)*r5;
    r3=T3(1:3,4)+Q13(1:3,1:3)*r4;
    r2=T2(1:3,4)+Q12(1:3,1:3)*r3;
    r1=T1(1:3,4)+Q11(1:3,1:3)*r2;

  %now they are transformed in the base referential:

    r1=r1;
    r2=Q11*r2;
    r3=Q12*r3;
    r4=Q13*r4;
    r5=Q14*r5;
    r6=Q15*r6;
%     r7=Q16*r7;

  %now we can get the Jacobian with the equation op Report TP2 Page 2
    J=zeros(6,6);
    J(1:6,1)=[e1;cross(e1,r1)];          J(1:6,2)=[e2;cross(e2,r2)];
    J(1:6,3)=[e3;cross(e3,r3)];          J(1:6,4)=[e4;cross(e4,r4)];
    J(1:6,5)=[e5;cross(e5,r5)];          J(1:6,6)=[e6;cross(e6,r6)];
```

Enrigistre.trj

```
function etat=enregistre_trj(trj, dT, filename)
% Updated to fit T3

%Expects a dT, a filename and a  nx6 matrix in the following format
%[  teta1_1 teta1_2 ...toolcommand1
%   teta2_1 teta2_2 ...toolcommand
%   teta3_1 teta3_2
%   teta4_1 teta4_2
%   teta5_1 teta5_2
%   teta6_1 teta6_2 ]  here with 6 tetas and 2 points


etat=0;
trj=trj';
[a, z]=size(trj);       % Dimensions of trj

if a==7

    fid=fopen(filename, 'w');  %trying to write the file
    if fid==0
            disp('Cant write the file:'); filename
        else
            fprintf(fid, '%10.0f ', z);      %Line numbers
            fprintf(fid, '%10.5f \n', dT);   %dT
            for i=1:z
            fprintf(fid, '%5.9f %5.9f %5.9f %5.9f %5.9f %5.9f %5.9f\n', trj(:,i));
            % writing the lines with the angles
            end
            fclose(fid);
            etat=1;
     end
 else
     disp('the matrix does not have 6 angles plus a tool command!');
 end
```

articulaire.m

```
function thetas=articulaire(t, dt, PF, QF, points)

disp('Calculating the articular path now');

for i=1:length(PF)
i
[t, label]=reach(PF(:, i), QF, t);  %Calculates the joint angles

% In case angles are multiples of pi we set it back
 for ti=1:6
     while t(ti)>(2*pi)
         t(ti)=t(ti)-(2*pi);
     end
   while t(ti)<(-2*pi)
         t(ti)=t(ti)+(2*pi);
     end
 end

thetas(i,:)=t;

label_all(i)=label;    %Writing the labels

end
trj=[thetas, ones(points, 1)]; % writing the new angles to trj
       %Storing it

%----------------------saving the path-----------------------------------
enregistre_trj(trj, dt, 'trajectoire_art.trj');% Save the partial track
%______________________________________________________________



end
```