IMPLEMENTATION OF IMPEDANCE CONTROL FOR A TELE-ECHOGRAPHIC SYSTEM

DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

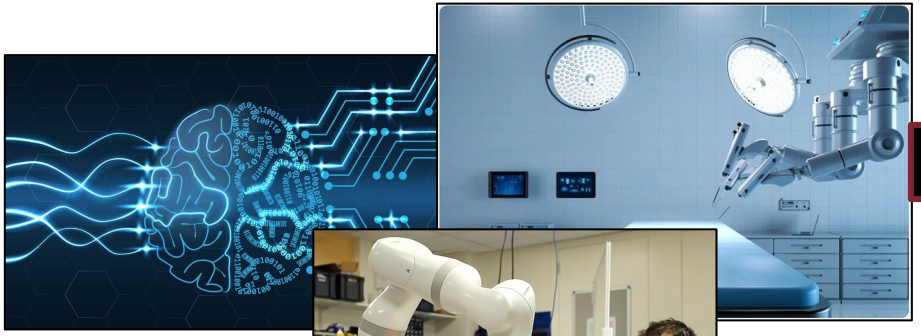
SAPIENZA UNIVERSITÀ DI ROMA

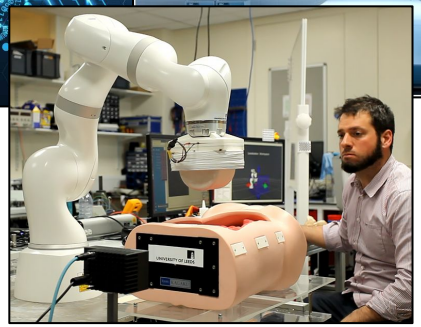
Caterina Borzillo - 1808187
Federica Cocci - 1802435
Damiano Gasperini - 1796880
Matteo Germano - 1807599

Medical Robotics - 01/04/2022

Introduction

Thanks to the **rapid development** of the Internet and of new technologies, such as **robotics**,



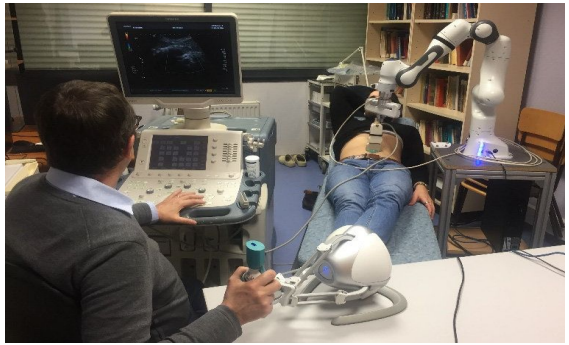


tele-medicine and in particular teleoperated systems had the opportunity to **spread worldwide**.

What is a tele-echographic system?

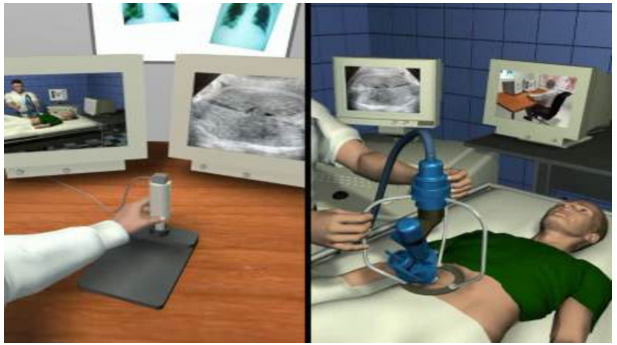
- It is a teleoperated system that is able to perform a **remote examination** which consists in the analysis and then transmission of **ultrasound video images** conducted by an expert doctor on a patient supported by a technician. Those systems are classified in:

- **short-distance:**



The master and slave systems located in the **same room**.

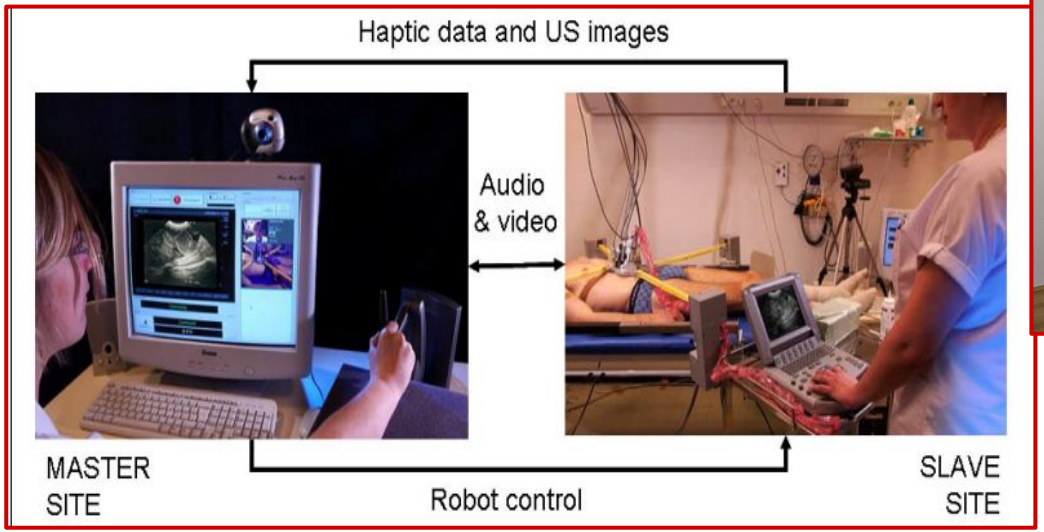
- **long-distance:**



The master and slave sites are **geographically separated**.

State of the art (1)

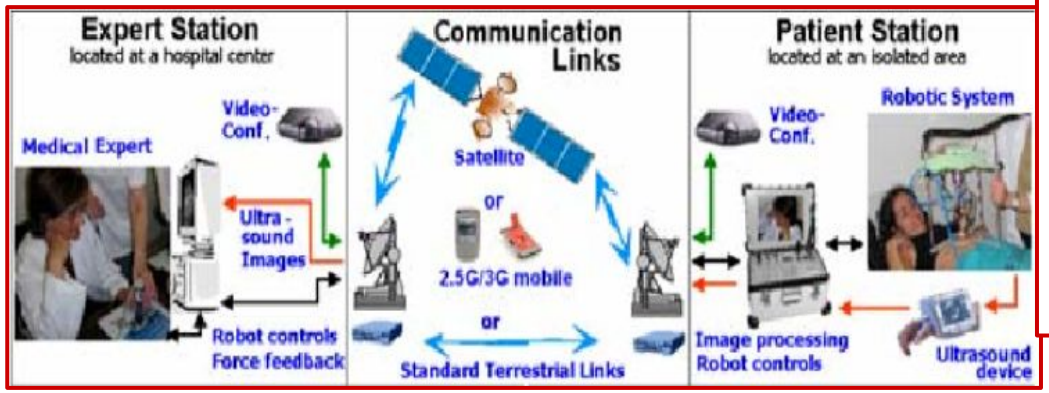
➔ **the TER system**



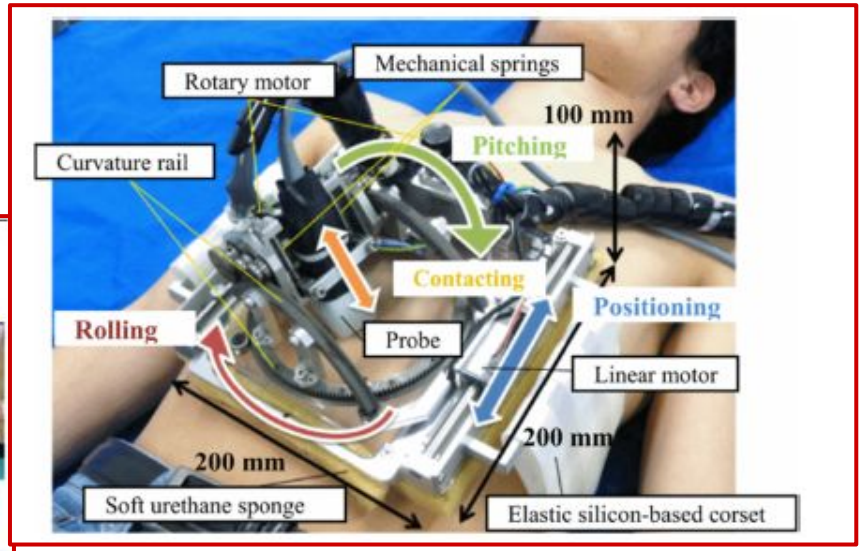
➔ **the MELODY system**

State of the art (2)

➔ **The OTELO system**



➔ **The FASTele system**



Task of the project

The simulated manipulator is equipped with an **ultrasound probe** which will be in contact with an **abdomen phantom**.

Our task: implementation of an **impedance control** for a manipulator with the **future goal** of remotely performing a commanded tele-echographic examination.

Impedance Control

- Interaction of the manipulator with the environment

- Small contact forces are desired in the interaction with a human body

- Two kind of impedance control:

    1. mechanical impedance
    2. linear impedance

Impedance Control

- Dynamic model:

$$u = B(q)y + n(q,\dot{q}) \quad \text{with} \quad n(q,\dot{q}) = C(q,\dot{q})\dot{q} + F\dot{q} + g(q)$$

- In the presence of end effector-forces

$$\ddot{q} = y - B^{-1}(q)J^{T}(q)h_e$$

- y can be chosen as

$$y = J_A^{-1}(q)M_d^{-1}\left(M_d\ddot{x}_d + K_D\dot{\tilde{x}} + K_P\tilde{x} - M_d\dot{J}_A(q,\dot{q})\dot{q}\right)$$

Impedance Control

- Second order differential kinematics

$$\boldsymbol{M}_d\ddot{\tilde{\boldsymbol{x}}} + \boldsymbol{K}_D\dot{\tilde{\boldsymbol{x}}} + \boldsymbol{K}_P\tilde{\boldsymbol{x}} = \boldsymbol{M}_d\boldsymbol{B}_A^{-1}(\boldsymbol{q})\boldsymbol{h}_A$$

- It establishes a relationship through a generalized mechanical impedance between the vector of forces and the vector of displacements in the operational space

- The system is coupled, contact force measure to resolve this problem

Impedance Control

- Choose

$$\boldsymbol{u} = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{y} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{h}_e$$

with

$$\boldsymbol{y} = \boldsymbol{J}_A^{-1}(\boldsymbol{q})\boldsymbol{M}_d^{-1}\left(\boldsymbol{M}_d\ddot{\boldsymbol{x}}_d + \boldsymbol{K}_D\dot{\tilde{\boldsymbol{x}}} + \boldsymbol{K}_P\tilde{\boldsymbol{x}} - \boldsymbol{M}_d\dot{\boldsymbol{J}}_A(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{h}_A\right)$$

under the assumption of error-free force measurements, yields

$$\boldsymbol{M}_d\ddot{\tilde{\boldsymbol{x}}} + \boldsymbol{K}_D\dot{\tilde{\boldsymbol{x}}} + \boldsymbol{K}_P\tilde{\boldsymbol{x}} = \boldsymbol{h}_A$$

KUKA framework (1)

- First attempt to resolve the task
- Developed using Visual Studio and V-REP and then CoppeliaSim

V-REP scenario

1. In order to implement the control law, we needed the torque control while in the given framework there was the position control

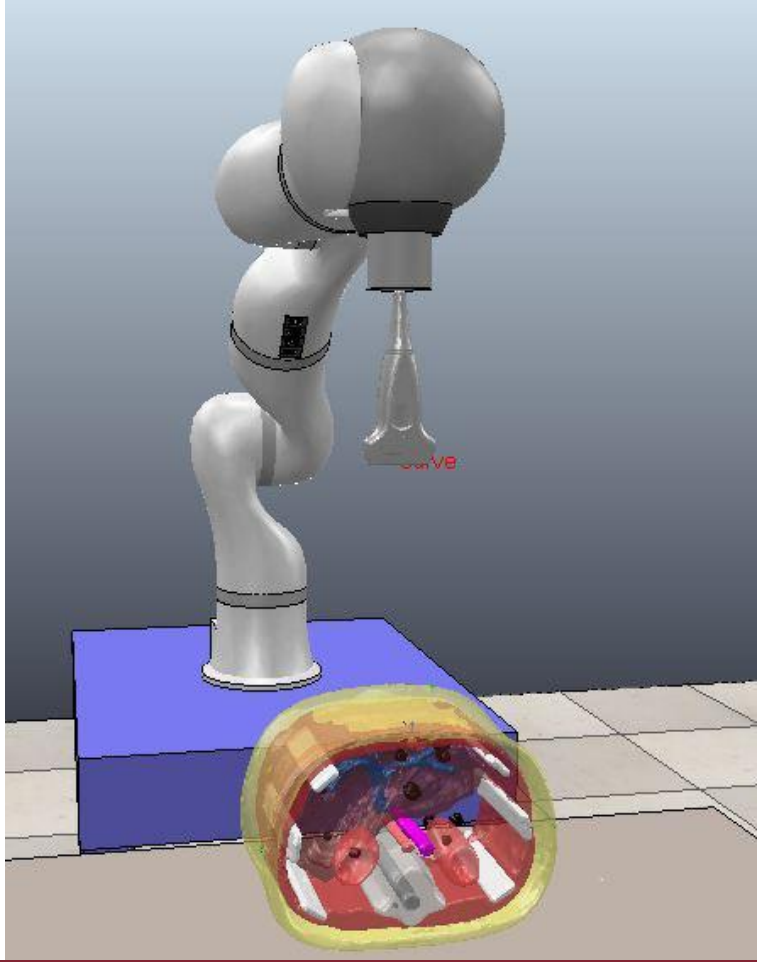
   → implementation of setJointTorque()

```
void VREPProxy::setJointTorque(const char* objName, const float& tau, const int& simport) {

  simxSetJointMaxForce(this->portIdMap[simport], this->simObjects[objName], abs(tau), simx_opmode_oneshot);
  if (tau >= 0) {
    //simxSetJointTargetPosition(this->portIdMap[simport], this->simObjects[objName], 1e10, simx_opmode_oneshot); // if CoppeliaSim joint control loop is enabled for this joint
      simxSetJointTargetVelocity(this->portIdMap[simport], this->simObjects[objName], 1e10, simx_opmode_oneshot); // if CoppeliaSim joint control loop is NOT enabled for this joint
  }
  else {
    //simxSetJointTargetPosition(this->portIdMap[simport], this->simObjects[objName], -1e10, simx_opmode_oneshot); // if CoppeliaSim joint control loop is enabled for this joint
      simxSetJointTargetVelocity(this->portIdMap[simport], this->simObjects[objName], -1e10, simx_opmode_oneshot); // if CoppeliaSim joint control loop is NOT enabled for this joint
  }
}
```

2. Actually in the function setJointTorque, a function called simxSetJointMaxForce was used but it was not compatible with V-REP version

   → change from V-REP to CoppeliaSim



KUKA framework (2)

CoppeliaSim scenario

1. Implementation of the control law

   → not working

2. Reduce the task to a regulation problem by using a dummy in the CoppeliaSim scene

   → randomic movement

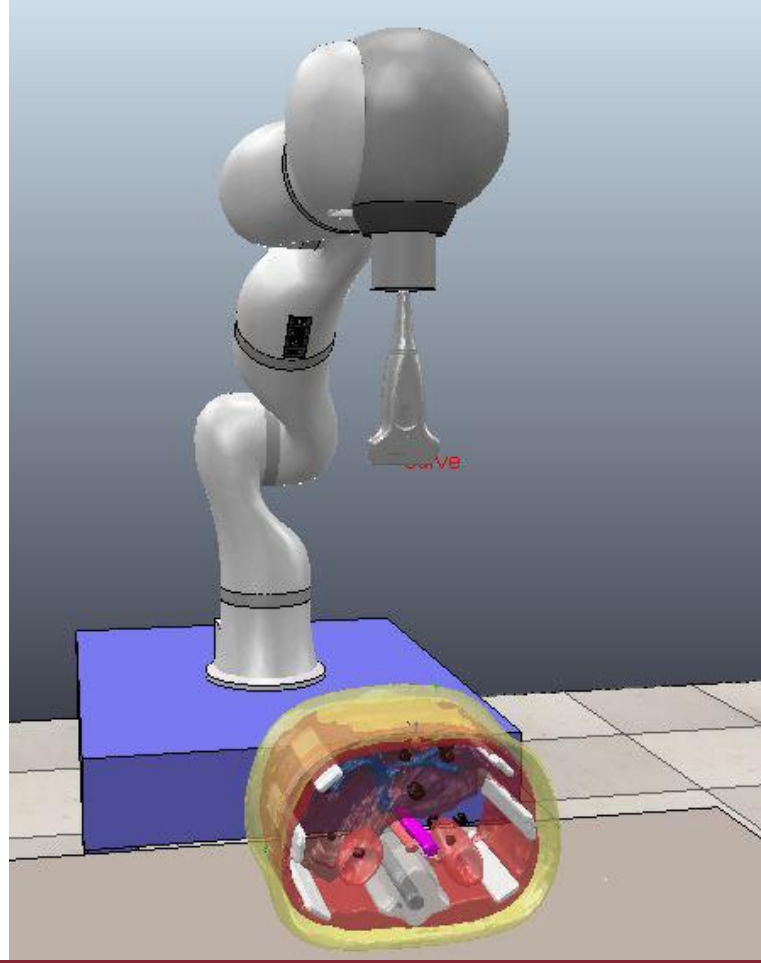
3. Analyse the evolution over time of commanded and desired torques by means of plots

   → two completely different behaviours

4. Implementation of the easiest control law: u = g

   → failure

→ WE SUPPOSE A PROBLEM WITH THE ALREADY IMPLEMENTED DYNAMIC

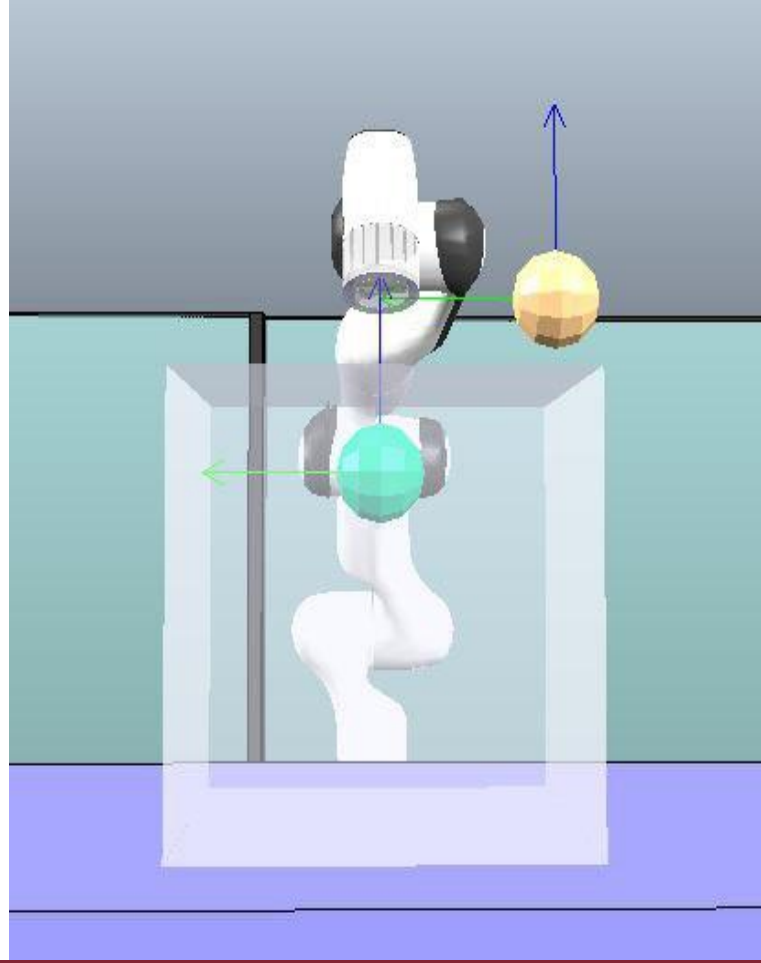


FRANKA framework (1)

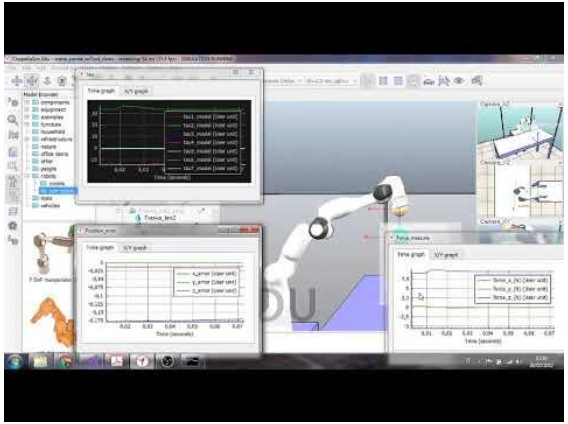
- Second attempt to resolve the task
- Developed using Visual Studio and CoppeliaSim

This time the scene in CoppeliaSim contains only the manipulator without the phantom, the end-effector/probe and the force sensor.

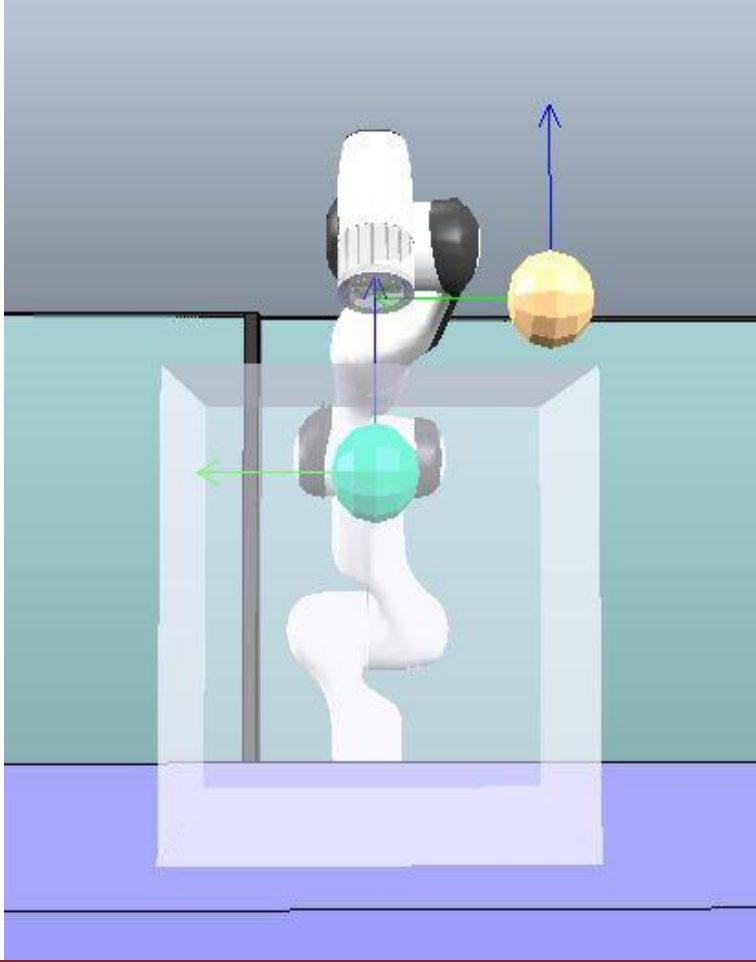
1. Implementation of the easiest case of torque control u = g
   → all works
   → step by step increase the complexity of the control law
2. Implementation of the regulation task using a dummy object
   → success
3. Addition of the force sensor and a table with a small cube on it
   → simulation of a contact force
4. Implementation of the regulation control law moving the dummy just below the cube surface
   → random movement (also using block gain K) in sensor/sensorless case



FRANKA framework (2)



In our opinion there could be several problems related to the CoppeliaSim scene or to the robot's dynamic model and moreover we noticed that in the received scene not all the links of the robot were dynamically enabled and respondable, introducing other possible errors.



FRANKA Framework with ROS

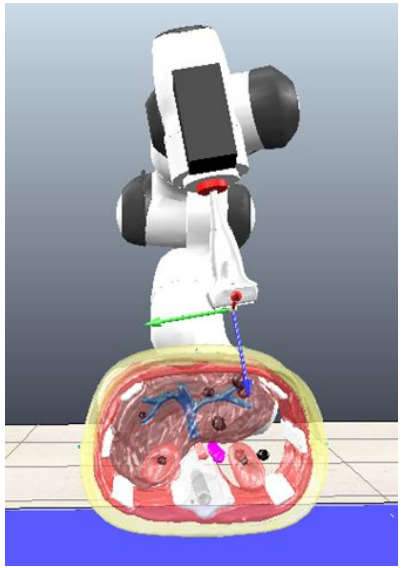
Main requirements

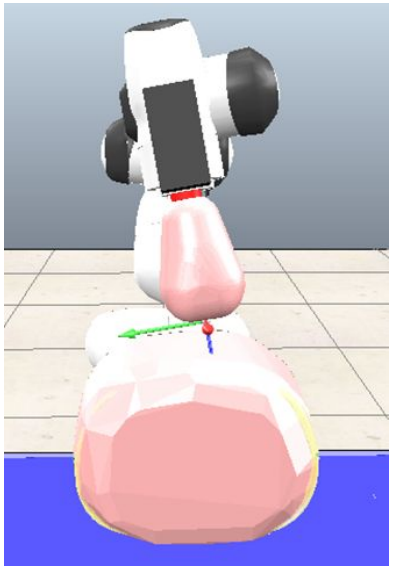
- ROS (melodic or noetic)
- CoppeliaSim
- visp_ros package
- Catkin tools

ViSP library

- It allows to control a real Panda robot from Franka Emika.
- visp_ros is an extension of ViSP library developed by Inria Rainbow team.
- In visp_ros we benefit from ROS features.
- Using visp_ros makes possible to simulate a Franka robot in position, velocity and force control.
- All the rendering and sensing is done through CoppeliaSim while the geometric and dynamic model is handled in visp_ros.

CoppeliaSim scene





- A script is associated to the FRANKA object in the scene, in which we initialize the handlers to manage different robot components, publishers and subscribers.

- Communication between visp_ros and CoppeliaSim is done using ROS.
- Some relevant topics:
  - /coppeliasim/franka/joint_state: measured positions, velocities and torques
  - /coppeliasim/franka/g0: gravity value
  - /coppeliasim/franka/flMe: position xyz, orientation xyzw
  - /coppeliasim/franka/tool/inertia: mass, center of mass, tool position xyz, inertia values
  - /fakeFCI/joint_state: commanded torques.

```
vpColVector q( 7, 0 ), dq( 7, 0 ), tau_d( 7, 0 ), C( 7, 0 ), F( 7, 0 ), tau_d0( 7, 0 ), tau_cmd( 7, 0 ),
    x_e( 6, 0 ), dx_e( 6, 0 ), dx_ed( 6, 0 ), ddx_ed( 6, 0 );
vpMatrix fJe( 6, 7 ), Ja( 6, 7 ), dJa( 6, 7 ), Ja_old( 6, 7 ), B( 7, 7 ), I7( 7, 7 ), Ja_pinv_B_t( 6, 7 );
vpColVector pose_err_norm( 2, 0 ), tau( 7, 0 );

std::cout << "Reading current joint position" << std::endl;
robot.getPosition( vpRobot::JOINT_STATE, q );
std::cout << "Initial joint position: " << q.t() << std::endl;

robot.setRobotState( vpRobot::STATE_FORCE_TORQUE_CONTROL );
robot.setCoppeliasimSyncMode( opt_coppeliasim_sync_mode );

vpHomogeneousMatrix fMed, fMed0;
fMed0 = robot.get_fMe();
fMed  = fMed0;

bool final_quit       = false;
bool first_time       = false;
bool start_trajectory = false;

vpMatrix K( 6, 6 ), D( 6, 6 ), edVf( 6, 6 );

double wp = 50;
double wo = 20;
double wz = 10;
//K.diag( { wp * wp, wp * wp, wp * wp, wo * wo, wo * wo, wo * wo } );
K.diag( { wp * wp, wp * wp, wz * wz , wp * wp, wo * wo, wo * wo } );
D.diag( { 2 * wp, 2 * wp, 2 * wp, 2 * wo, 2 * wo, 2 * wo } );
I7.eye();
//std::cout << "K: " << K<< std::endl;

double mu = 4;
double dt = 0;

double time_start_trajectory, time_prev, time_cur;
```

Code (0)

```
170     double delay_before_trajectory = 0.5; // Start sinusoidal joint trajectory after this delay in [s]
171
172     // Control loop
173     while ( !final_quit )
174     {
175       time_cur = robot.getCoppeliasimSimulationTime();
176
177       robot.getPosition( vpRobot::JOINT_STATE, q );
178       robot.getVelocity( vpRobot::JOINT_STATE, dq );
179       robot.getMass( B );
180       robot.getCoriolis( C );
181       robot.getFriction( F );
182       robot.get_fJe( fJe );
183       robot.getForceTorque( vpRobot::JOINT_STATE, tau );
184       //std::cout << "fJe: " << fJe << std::endl;
185       //std::cout << "F: " << F << std::endl;
186       std::cout << "q: " << q.t() << std::endl;
187
188       if ( time_cur < delay_before_trajectory )
189       {
190         time_start_trajectory = time_cur; // To ensure exp() = 1
191         first_time            = true;
192       }
193       else if ( !start_trajectory ) // After the delay we start joint trajectory
194       {
195         time_start_trajectory = time_cur;
196         start_trajectory      = true;
197       }
198
199       // Compute Cartesian trajectories
200       fMed[1][3] = fMed0[1][3] + ( start_trajectory ? ( 0.05 * sin( 2 * M_PI * 0.3 * ( time_cur - time_start_trajectory  ) ) ) : 0 );
201       fMed[2][3] = fMed0[2][3] - ( start_trajectory ? 7.5446e-02 : 0);
202
203       dx_ed[1] = ( start_trajectory ? (2 * M_PI * 0.3 * 0.05 * cos( 2 * M_PI * 0.3 * ( time_cur - time_start_trajectory  ) ) ) : 0) ;
204       dx_ed[2] = 0;
```

Code (1)

```
ddx_ed[1] = ( start_trajectory ? (-2 * M_PI * 0.3 * 2 * M_PI * 0.3 * 0.05 * sin( 2 * M_PI * 0.3 * ( time_cur - time_start_trajectory  ) ) ) : 0) ;
ddx_ed[2] = 0;

edVf.insert( fMed.getRotationMatrix().t(), 0, 0 );
edVf.insert( fMed.getRotationMatrix().t(), 3, 3 );


x_e = (vpColVector)vpPoseVector( fMed.inverse() * robot.get_fMe() ); // edMe
Ja  = Ta( fMed.inverse() * robot.get_fMe() ) * edVf * fJe;

dx_e = Ta( fMed.inverse() * robot.get_fMe() ) * edVf * ( dx_ed - fJe * dq );

//std::cout << "x_e: " << x_e  << std::endl;
//std::cout << "Ja: " << Ja  << std::endl;
//std::cout << "dx_e: " << dx_e  << std::endl;
//std::cout << "ddx_ed: " << ddx_ed  << std::endl;

dt = time_cur - time_prev;

if ( dt != 0 )
{
  dJa = ( Ja - Ja_old ) / dt;
}
else
{
  dJa = 0;
}
Ja_old = Ja;

Ja_pinv_B_t = ( Ja * B.inverseByCholesky() * Ja.t() ).inverseByCholesky() * Ja * B.inverseByCholesky();

// Compute the control law
tau_d = B * Ja.pseudoInverse() * ( -K * ( x_e ) + D * (dx_e)-dJa * dq + ddx_ed ) + C + F -
        ( I7 - Ja.t() * Ja_pinv_B_t ) * B * dq * 100;
//std::cout << "tau_d: " << tau_d  << std::endl;
```

Code (2)

Code (3)

```
235     Ja_pinv_B_t = ( Ja * B.inverseByCholesky() * Ja.t() ).inverseByCholesky() * Ja * B.inverseByCholesky();
236
237     // Compute the control law
238     tau_d = B * Ja.pseudoInverse() * ( -K * ( x_e ) + D * (dx_e)-dJa * dq + ddx_ed ) + C + F -
239           ( I7 - Ja.t() * Ja_pinv_B_t ) * B * dq * 100;
240     //std::cout << "tau_d: " << tau_d  << std::endl;
241
242
243     if ( first_time )
244     {
245       tau_d0 = tau_d;
246     }
247
248     tau_cmd = tau_d - tau_d0 * std::exp( -mu * ( time_cur - time_start_trajectory ) );
249     //std::cout << "tau_cmd: " << tau_cmd  << std::endl;
250
251     robot.setForceTorque( vpRobot::JOINT_STATE, tau_cmd );
```

Results & Conclusions