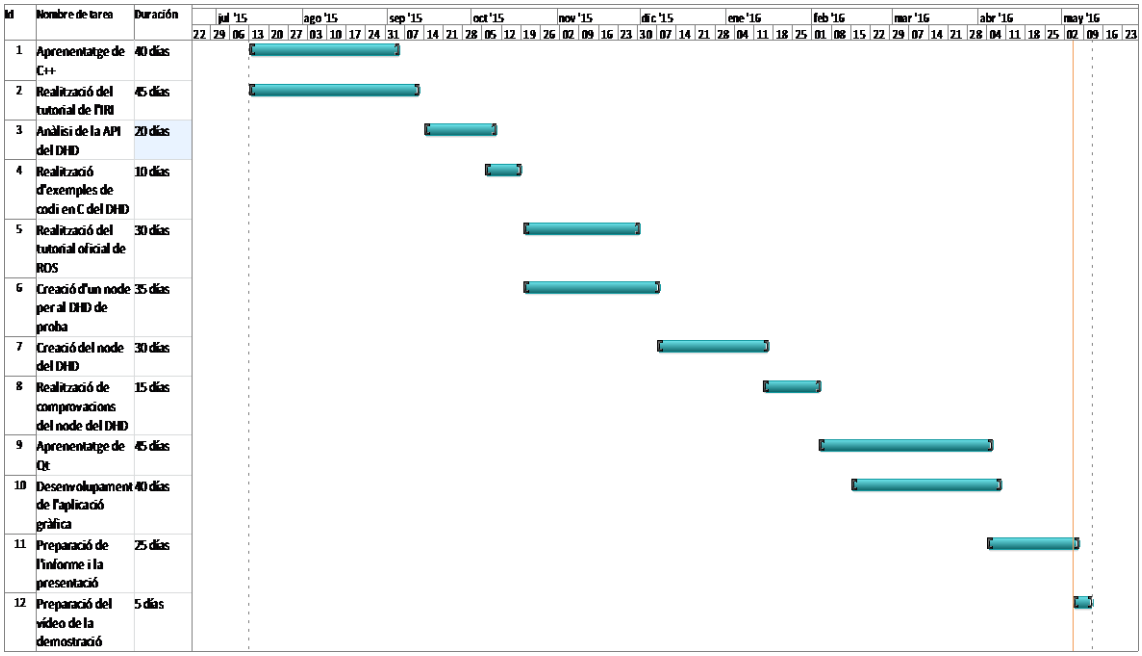
Contingut

1. Annex A: Gestió del Projecte

1.1. Planificació del Projecte

A continuació es mostra un Diagrama de Gantt amb les diferents etapes planificades per al projecte.



1.2. Cost del Projecte

Per poder fer un pressupost del cost del projecte, cal entendre quin ha estat el treball realitzat concretament.
El projecte no consisteix en la creació de tot un entorn de treball a escala de hardware, sinó l'adaptació d'un dispositiu ja existent al laboratori de l'Institut de Robòtica i Informàtica Industrial (el dispositiu hàptic) en un entorn de software que els altres elements del laboratori fan servir (ROS) i el redisseny d'una aplicació gràfica que pugui interactuar amb els altres dispositius fent servir ROS.
Tenint això en compte, es pot afirmar que el cost monetari del projecte és enterament la retribució que suposa l'esforç humà d'adaptació del DHD i l'aplicació gràfica al nou sistema informàtic.
Considerant que el treball diari de l'autor del projecte ha estat aproximadament de 3 h diàries durant 7 mesos útils de treball (traient caps de setmana i mesos de vacances o dedicats a altres tasques acadèmiques) i que el preu que l'autor cobra per hora, ja que és un enginyer junior, es pot fer la següent estimació:

Element	Quantitat	Preu per unitat	Preu
Hores de treball de l'autor	420 h	7 €/h	2940 €
Preu total			2940 €

Per tant, el cost del projecte s'estima entorn dels 2.940 €.

Si en canvi es considera tot el material fet servir per poder fer servir el sistema partint de zero, cal incloure tot el cost de hardware que això implica. Amb preu de mercat actual i segons la informació obtinguda dels llocs web oficials dels fabricants, una bona aproximació per una instal·lació mínimament moderna podria ser la següent:

Element	Quantitat	Preu per unitat	Preu
Hores de treball de l'autor	420 h	7 €/h	2.940 €
Ordinador de treball	1	1.000 €/u	1.000 €
Delta Haptic Device Force Dimension Delta 6	1	45.000 €/u	45.000 €
Stäubli 6-axis TX90XL	1	56.250 €/u	56.250 €
Schunk FTD Gamma	1	1.000 €/u	1.000 €
Preu total			106.190 €

El cost en aquest cas és de 106.190 €.

Cal notar, però, que no es contemplen les hores de treball humà necessàries per la correcta instal·lació de tot el hardware necessari ni les hores necessàries per crear els nodes de ROS del braç robòtic o del sensor de forces, per tant el preu total acabaria sent inclús superior.

Existeix també un mercat de robots de segona mà que podria abaratir considerablement (entorn al 60%) el preu imputable a la compra del hardware necessari.

1.3. Impacte mediambiental

A causa de la naturalesa del projecte, on no s'ha de construir cap sistema, sinó implementar aparells ja existents en un entorn determinat de software, és evident que l'impacte mediambiental del projecte és nul.

Es podria fer un estudi de l'impacte que comporta la utilització del sistema de telecomandament, però ja que és una instal·lació d'investigació i no de producció a gran escala, la conclusió continua sent que l'impacte ambiental és gairebé zero.

De fet, integrar equips relativament vells en un entorn de
ROS, si és possible, pot suposar l'allargament de la vida útil d'aquests, amb la qual cosa l'impacte mediambiental és més aviat positiu.

2. Annex B: Codi del Driver del DHD

2.1. delta_haptic_device.h

```
#ifndef _DELTA_HAPTIC_DEVICE_H
#define _DELTA_HAPTIC_DEVICE_H

#include <iostream>
#include <dhdc.h>

class CDelta_Haptic_Device
{

  public:
    CDelta_Haptic_Device();
    ~CDelta_Haptic_Device();
    bool ON;
    bool status_ok;

    /* retrieves the position p (in metres) and the angle o (in
degrees) of the End Effector */
    int getPositionAndOrientation (double& px, double& py, double& pz,
double& oa, double& ob, double& oc);

    /* retrieves the estimated instantaneous linear velocity v (in
m/s)  */
    int getLinearVelocity (double& vx, double& vy, double& vz);

    /* starts the device, enables expert mode, gets SDKversion and
enables forces to be applied
     *
     * returns 0 if the device succesfuly does all of this
     * returns -1 and a message error otherwise
     */
    int open();

    /* sets force f (in N) and torque t (in Nm) to the end effector*/
    int setForceAndTorque (double fx, double fy, double fz, double tx,
double ty, double tz);

};

#endif
```

2.2. delta_haptic_device.cpp

```
#include "delta_haptic_device.h"
#include <stdexcept>
#include <typeinfo>
#include <math.h>

const double fx_max = 10;
const double fy_max = 10;
const double fz_max = 10;

const double fx_min = -10;
const double fy_min = -10;
const double fz_min = -10;

const double tx_max = 0.2;
const double ty_max = 0.2;
const double tz_max = 0.2;

const double tx_min = -0.2;
const double ty_min = -0.2;
const double tz_min = -0.2;



CDelta_Haptic_Device::CDelta_Haptic_Device()
{
  this->ON = false;
  this->status_ok = false;

  try
  {
    open();
  }
  catch(const std::exception& e)
  {
    std::cerr << e.what()  << std::endl;
  }
}

CDelta_Haptic_Device::~CDelta_Haptic_Device()
{
  this->ON = false;
  dhdStop();
  dhdDisableExpertMode();
  dhdClose();
}

int CDelta_Haptic_Device::getPositionAndOrientation (double& px,
double& py, double& pz, double& oa, double& ob, double& oc)
{
  if (ON && status_ok)
  {
    try
    {
      if (dhdGetPositionAndOrientationDeg (&px, &py, &pz, &oa, &ob,
&oc) == -1)
        throw std::runtime_error(std::string("Error:
dhdGetPositionAndOrientationDeg ")+dhdErrorGetLastStr());
    }
```

```
    catch(const std::exception& e)
    {
      std::cerr << e.what()  << std::endl;
    }
  }
  else
    return -1;
  return 0;
}

int CDelta_Haptic_Device::getLinearVelocity (double& vx, double& vy,
double& vz)
{
  if (ON && status_ok)
  {
    try
    {
      if (dhdGetLinearVelocity(&vx, &vy, &vz) == -1)
        throw std::runtime_error(std::string("Error:
dhdGetLinearVelocity ")+dhdErrorGetLastStr());
    }
    catch(const std::exception& e)
    {
      std::cerr << e.what()  << std::endl;
    }
  }
  else
    return -1;
  return 0;
}

int CDelta_Haptic_Device::open()
{
  int major, minor, release, revision;
  double version;
  ushort serial;
  std::string sys_name;
  if (!this->ON)
    {
    if (dhdOpen() < 0)
      {
      std::string msg = std::string("Error: ")+dhdErrorGetLastStr();
      throw std::runtime_error(msg);
      return -1;
      }

    else
      {
      dhdEnableExpertMode();
      dhdEnableForce (DHD_ON);
      dhdGetSDKVersion(&major, &minor, &release, &revision);
      dhdGetSerialNumber(&serial);
      dhdGetVersion(&version);
      sys_name = dhdGetSystemName();
      double resta = version-2.1;
      double resta1 = serial-2;

      /*std::cout << "SDK Version: " << major << ", " << minor << ", "
<< release << ", " << revision << std::endl;
      std::cout << "DHD Version: " << version << std::endl;
      std::cout << "DHD ID: " << dhdGetDeviceID() << std::endl;
```

```
      std::cout << "DHD Serial Number: " << serial << std::endl;
      std::cout << "DHD System Name: " << dhdGetSystemName() <<
std::endl;
      std::cout << "DHD System Type: " << dhdGetSystemType() <<
std::endl;*/
      if (major == 3 && minor == 5 && release == 3 && revision == 2652
&& fabs(resta) <= 0.1 && fabs(resta1) <= 0.1 && sys_name == "delta.6"
&& dhdGetSystemType() == 64)
      {
        this->status_ok = true;
      }
      else
        throw std::runtime_error("Error: Delta Haptic Device
configuration failed. ");

      this->ON = true;
      return 0;
      }
    }
  else
    throw std::runtime_error("Error: dhdOpen() function called twice!
");
}

int CDelta_Haptic_Device::setForceAndTorque(double fx, double fy,
double fz, double tx, double ty, double tz)
{
  if (fx > fx_max)
    fx = fx_max;
  if (fy > fy_max)
    fy = fy_max;
  if (fz > fz_max)
    fz = fz_max;

  if (fx < fx_min)
    fx = fx_min;
  if (fy < fy_min)
    fy = fy_min;
  if (fz < fz_min)
    fz = fz_min;

  if (tx > tx_max)
    tx = tx_max;
  if (ty > ty_max)
    ty = ty_max;
  if (tz > tz_max)
    tz = tz_max;

  if (tx < tx_min)
    tx = tx_min;
  if (ty < ty_min)
    ty = ty_min;
  if (tz < tz_min)
    tz = tz_min;

  if (ON && status_ok)
  {
    try
    {
      if (dhdSetForceAndTorque(fx, fy, fz, tx, ty, tz) == -1)
```

```
        throw std::runtime_error(std::string("Error:
dhdSetForceAndTorque ")+dhdErrorGetLastStr());
    }
    catch(const std::exception& e)
    {
      std::cerr << e.what()  << std::endl;
    }
  }
  else
    return -1;
  return 0;
}
```

2.3. delta_haptic_device_test.cpp

```
#include "delta_haptic_device.h"
#include <iomanip>

const double ACTION_TIME = 0.4;  //how long does the action lasts

int main(int argc, char *argv[])
{
  double px, py, pz;
  double oa, ob, oc;
  double vx, vy, vz;
  double t1,t0 = dhdGetTime(), ta, tb;
  int done;

  done = 0;
  CDelta_Haptic_Device simulation;

  if (simulation.ON)
  {

    std::cout << "Choose one mode:" << std::endl;
    std::cout << "Press '1' to impulse the End Effector on the x axis" << std::endl;
    std::cout << "Press '2' to impulse the End Effector on the y axis" << std::endl;
    std::cout << "Press '3' to impulse the End Effector on the z axis" << std::endl;
    std::cout << "Press '4' to totate the End Effector over the x axis" << std::endl;
    std::cout << "Press '5' to totate the End Effector over the y axis" << std::endl;
    std::cout << "Press '6' to totate the End Effector over the z axis" << std::endl;
    std::cout << "Press 'q' to quit\n\n" << std::endl;

    while (done==0)
    {
      simulation.setForceAndTorque(0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
      simulation.getPositionAndOrientation(px, py, pz, oa, ob, oc);
      simulation.getLinearVelocity(vx, vy, vz);
      t1 = dhdGetTime ();
      if ((t1-t0) > 0.1)
      {
        t0   = t1;
        std::cout << std::fixed;
        std::cout << std::setprecision(3);
        std::cout << "P: " << px << ", " << py << ", " << pz << " (m)";
        std::cout << " O: " << oa << ", " << ob << ", " << oc << " (°)";
        std::cout << " V: " << vx << ", " << vy << ", " << vz << " (m/s) \r";

        if (dhdKbHit())
        {
          ta = dhdGetTime();
          tb = dhdGetTime();
          switch (dhdKbGet())
          {
        case '1': while ((tb-ta) < ACTION_TIME)
                  {
                        simulation.setForceAndTorque(3.5, 0.0, 0.0, 0.0, 0.0, 0.0);
                    tb = dhdGetTime();
                    };  break;
            case '2': while ((tb-ta) < ACTION_TIME)
                  {
                    simulation.setForceAndTorque(0.0, 2.5, 0.0, 0.0, 0.0, 0.0);
                    tb = dhdGetTime();
                    }; break;
```

```
            case '3': while ((tb-ta) < ACTION_TIME)
                  {
                      simulation.setForceAndTorque(0.0, 0.0, 3.0, 0.0, 0.0, 0.0);
                      tb = dhdGetTime();
                      }; break;
            case '4': while ((tb-ta) < ACTION_TIME)
                  {
                      simulation.setForceAndTorque(0.0, 0.0, 0.0, 0.1, 0.0, 0.0);
                      tb = dhdGetTime();
                      }; break;
            case '5': while ((tb-ta) < ACTION_TIME)
                  {
                      simulation.setForceAndTorque(0.0, 0.0, 0.0, 0.0, 0.1, 0.0);
                      tb = dhdGetTime();
                      }; break;
            case '6': while ((tb-ta) < ACTION_TIME)
                  {
                      simulation.setForceAndTorque(0.0, 0.0, 0.0, 0.0, 0.0, 0.1);
                      tb = dhdGetTime();
                      }; break;

            case 'q': done = 1; break;
          }
        }
      }
    }
  }

  dhdClose();
}
```

3. Annex C: Codi del Node del DHD

3.1. CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(iri_delta_haptic_device)

## Find catkin macros and libraries
find_package(catkin REQUIRED)
# *****************************************************************
#                 Add catkin additional components here
# *****************************************************************
find_package(catkin REQUIRED COMPONENTS iri_base_driver std_msgs
geometry_msgs message_generation)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)

# *****************************************************************
#           Add system and labrobotica dependencies here
# *****************************************************************
find_package(delta_haptic_device REQUIRED)

# *****************************************************************
#           Add topic, service and action definition here
# *****************************************************************
## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
add_service_files(
  FILES
  ForceAndTorque.srv
)

## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

## Generate added messages, services and actions with any dependencies
listed here
```

```
generate_messages(
  DEPENDENCIES
  geometry_msgs
  std_msgs  # Or other packages containing msgs
)

# ******************************************************************************
#                  Add the dynamic reconfigure file
# ******************************************************************************
generate_dynamic_reconfigure_options(cfg/DeltaHapticDevice.cfg)

# ******************************************************************************
#                  Add run time dependencies here
# ******************************************************************************
catkin_package(
#  INCLUDE_DIRS
#  LIBRARIES
# ******************************************************************************
#            Add ROS and IRI ROS run time dependencies
# ******************************************************************************
 CATKIN_DEPENDS iri_base_driver geometry_msgs std_msgs
# ******************************************************************************
#      Add system and labrobotica run time dependencies here
# ******************************************************************************
 DEPENDS delta_haptic_device
)

############
## Build ##
############

# ******************************************************************************
#                    Add the include directories
# ******************************************************************************
include_directories(include)
include_directories(${catkin_INCLUDE_DIRS})
include_directories(${delta_haptic_device_INCLUDE_DIR})

## Declare a cpp library
# add_library(${PROJECT_NAME} <list of source files>)

## Declare a cpp executable
add_executable(${PROJECT_NAME} src/delta_haptic_device_driver.cpp
src/delta_haptic_device_driver_node.cpp)

# ******************************************************************************
#                    Add the libraries
# ******************************************************************************
target_link_libraries(${PROJECT_NAME} ${catkin_LIBRARIES})
```

```
target_link_libraries(${PROJECT_NAME} ${delta_haptic_device_LIBRARY})
target_link_libraries(${PROJECT_NAME}
/usr/local/lib/iridrivers/libhaptic.a /home/jlaplaza/soft/foreign/sdk-
3.5.3/lib/release/lin-x86_64-gcc/libdhd.a -lusb-1.0 -pthread)

# ****************************************************************
#               Add message headers dependencies
# ****************************************************************
# add_dependencies(${PROJECT_NAME}
<msg_package_name>_generate_messages_cpp)
add_dependencies(${PROJECT_NAME} geometry_msgs_generate_messages_cpp)
add_dependencies(${PROJECT_NAME} geometry_msgs_generate_messages_cpp
std_msgs_generate_messages_cpp)
# ****************************************************************
#               Add dynamic reconfigure dependencies
# ****************************************************************
add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS})
```

3.2. package.xml

```
<?xml version="1.0"?>
<package>
  <name>iri_delta_haptic_device</name>
  <version>0.0.0</version>
  <description>The iri_delta_haptic_device package</description>

  <maintainer
email="javier.laplaza.galindo@gmail.com">javierlaplaza</maintainer>

  <license>BSD</license>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>iri_base_driver</build_depend>
  <build_depend>geometry_msgs</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>delta_haptic_device</build_depend>

  <run_depend>iri_base_driver</run_depend>
  <run_depend>geometry_msgs</run_depend>
  <run_depend>delta_haptic_device</run_depend>
  <run_depend>std_msgs</run_depend>

  <export>
  </export>
</package>
```

3.3. delta_haptic_device_driver.h

```
// Copyright (C) 2010-2011 Institut de Robotica i Informatica
Industrial, CSIC-UPC.
// Author
// All rights reserved.
//
// This file is part of iri-ros-pkg
// iri-ros-pkg is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
published by
// the Free Software Foundation, either version 3 of the License, or
// at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
License
// along with this program.  If not, see
<http://www.gnu.org/licenses/>.
//
// IMPORTANT NOTE: This code has been generated through a script from
the
// iri_ros_scripts. Please do NOT delete any comments to guarantee the
correctness
// of the scripts. ROS topics can be easly add by using those scripts.
Please
// refer to the IRI wiki page for more information:
// http://wikiri.upc.es/index.php/Robotics_Lab

#ifndef _delta_haptic_device_driver_h_
#define _delta_haptic_device_driver_h_

#include <iri_base_driver/iri_base_driver.h>
#include <iri_delta_haptic_device/DeltaHapticDeviceConfig.h>
#include "delta_haptic_device.h"

//include delta_haptic_device_driver main library

/**
 * \brief IRI ROS Specific Driver Class
 *
 * This class inherits from the IRI Base class IriBaseDriver, which
provides the
 * guidelines to implement any specific driver. The IriBaseDriver
class offers an
```

```
 * easy framework to integrate functional drivers implemented in C++
with the
 * ROS driver structure. ROS driver_base state transitions are already
managed
 * by IriBaseDriver.
 *
 * The DeltaHapticDeviceDriver class must implement all specific
driver requirements to
 * safetely open, close, run and stop the driver at any time. It also
must
 * guarantee an accessible interface for all driver's parameters.
 *
 * The DeltaHapticDeviceConfig.cfg needs to be filled up with those
parameters suitable
 * to be changed dynamically by the ROS dyanmic reconfigure
application. The
 * implementation of the CIriNode class will manage those parameters
through
 * methods like postNodeOpenHook() and reconfigureNodeHook().
 *
 */
class DeltaHapticDeviceDriver : public iri_base_driver::IriBaseDriver
{
  private:
    // private attributes and methods
    /**
    * \brief starting operation
    */
    CDelta_Haptic_Device sim_;

  public:
   /**
    * \brief define config type
    *
    * Define a Config type with the DeltaHapticDeviceConfig. All
driver implementations
    * will then use the same variable type Config.
    */
    typedef iri_delta_haptic_device::DeltaHapticDeviceConfig Config;

   /**
    * \brief config variable
    *
    * This variable has all the driver parameters defined in the cfg
config file.
    * Is updated everytime function config_update() is called.
    */
    Config config_;
```

```
    /**
     * \brief constructor
     *
     * In this constructor parameters related to the specific driver
can be
     * initalized. Those parameters can be also set in the openDriver()
function.
     * Attributes from the main node driver class IriBaseDriver such as
loop_rate,
     * may be also overload here.
     */
    DeltaHapticDeviceDriver(void);

    /**
     * \brief open driver
     *
     * In this function, the driver must be openned. Openning errors
must be
     * taken into account. This function is automatically called by
     * IriBaseDriver::doOpen(), an state transition is performed if
return value
     * equals true.
     *
     * \return bool successful
     */
    bool openDriver(void);

    /**
     * \brief close driver
     *
     * In this function, the driver must be closed. Variables related
to the
     * driver state must also be taken into account. This function is
automatically
     * called by IriBaseDriver::doClose(), an state transition is
performed if
     * return value equals true.
     *
     * \return bool successful
     */
    bool closeDriver(void);

    /**
     * \brief start driver
     *
     * After this function, the driver and its thread will be started.
The driver
     * and related variables should be properly setup. This function is
```

```
     * automatically called by IriBaseDriver::doStart(), an state
transition is
     * performed if return value equals true.
     *
     * \return bool successful
     */
     bool startDriver(void);

    /**
     * \brief stop driver
     *
     * After this function, the driver's thread will stop its
execution. The driver
     * and related variables should be properly setup. This function is
     * automatically called by IriBaseDriver::doStop(), an state
transition is
     * performed if return value equals true.
     *
     * \return bool successful
     */
     bool stopDriver(void);

    /**
     * \brief config update
     *
     * In this function the driver parameters must be updated with the
input
     * config variable. Then the new configuration state will be stored
in the
     * Config attribute.
     *
     * \param new_cfg the new driver configuration state
     *
     * \param level level in which the update is taken place
     */
     void config_update(Config& new_cfg, uint32_t level=0);

     // here define all delta_haptic_device_driver interface methods to
retrieve and set
     // the driver parameters

    /**
     * \brief Destructor
     *
     * This destructor is called when the object is about to be
destroyed.
     *
     */
     ~DeltaHapticDeviceDriver(void);
```

```
    /* retrieves the position p (in metres) and the angle o (in
degrees) of the End Effector */
    int getPositionAndOrientation (double& px, double& py, double& pz,
double& oa, double& ob, double& oc);

    /* retrieves the estimated instantaneous velocity v (in m/s) */
    int getLinearVelocity (double& vx, double& vy, double& vz);

    /* sets force f (in N) and torque t (in Nm) to the end effector*/
    int setForceAndTorque (double fx, double fy, double fz, double tx,
double ty, double tz);

};

#endif
```

3.4. delta_haptic_device_driver.cpp

```
// Copyright (C) 2010-2011 Institut de Robotica i Informatica
Industrial, CSIC-UPC.
// Author
// All rights reserved.
//
// This file is part of iri-ros-pkg
// iri-ros-pkg is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
published by
// the Free Software Foundation, either version 3 of the License, or
// at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
License
// along with this program.  If not, see
<http://www.gnu.org/licenses/>.
//
// IMPORTANT NOTE: This code has been generated through a script from
the
// iri_ros_scripts. Please do NOT delete any comments to guarantee the
correctness
// of the scripts. ROS topics can be easly add by using those scripts.
Please
// refer to the IRI wiki page for more information:
// http://wikiri.upc.es/index.php/Robotics_Lab

#ifndef _delta_haptic_device_driver_h_
#define _delta_haptic_device_driver_h_

#include <iri_base_driver/iri_base_driver.h>
#include <iri_delta_haptic_device/DeltaHapticDeviceConfig.h>
#include "delta_haptic_device.h"

//include delta_haptic_device_driver main library

/**
 * \brief IRI ROS Specific Driver Class
 *
 * This class inherits from the IRI Base class IriBaseDriver, which
provides the
 * guidelines to implement any specific driver. The IriBaseDriver
class offers an
```

```
 * easy framework to integrate functional drivers implemented in C++
with the
 * ROS driver structure. ROS driver_base state transitions are already
managed
 * by IriBaseDriver.
 *
 * The DeltaHapticDeviceDriver class must implement all specific
driver requirements to
 * safetely open, close, run and stop the driver at any time. It also
must
 * guarantee an accessible interface for all driver's parameters.
 *
 * The DeltaHapticDeviceConfig.cfg needs to be filled up with those
parameters suitable
 * to be changed dynamically by the ROS dyanmic reconfigure
application. The
 * implementation of the CIriNode class will manage those parameters
through
 * methods like postNodeOpenHook() and reconfigureNodeHook().
 *
 */
class DeltaHapticDeviceDriver : public iri_base_driver::IriBaseDriver
{
  private:
    // private attributes and methods
    /**
    * \brief starting operation
    */
    CDelta_Haptic_Device sim_;

  public:
   /**
    * \brief define config type
    *
    * Define a Config type with the DeltaHapticDeviceConfig. All
driver implementations
    * will then use the same variable type Config.
    */
    typedef iri_delta_haptic_device::DeltaHapticDeviceConfig Config;

   /**
    * \brief config variable
    *
    * This variable has all the driver parameters defined in the cfg
config file.
    * Is updated everytime function config_update() is called.
    */
    Config config_;
```

```
    /**
     * \brief constructor
     *
     * In this constructor parameters related to the specific driver
can be
     * initalized. Those parameters can be also set in the openDriver()
function.
     * Attributes from the main node driver class IriBaseDriver such as
loop_rate,
     * may be also overload here.
     */
    DeltaHapticDeviceDriver(void);

    /**
     * \brief open driver
     *
     * In this function, the driver must be openned. Openning errors
must be
     * taken into account. This function is automatically called by
     * IriBaseDriver::doOpen(), an state transition is performed if
return value
     * equals true.
     *
     * \return bool successful
     */
    bool openDriver(void);

    /**
     * \brief close driver
     *
     * In this function, the driver must be closed. Variables related
to the
     * driver state must also be taken into account. This function is
automatically
     * called by IriBaseDriver::doClose(), an state transition is
performed if
     * return value equals true.
     *
     * \return bool successful
     */
    bool closeDriver(void);

    /**
     * \brief start driver
     *
     * After this function, the driver and its thread will be started.
The driver
     * and related variables should be properly setup. This function is
```

```
     * automatically called by IriBaseDriver::doStart(), an state
transition is
     * performed if return value equals true.
     *
     * \return bool successful
     */
     bool startDriver(void);

    /**
     * \brief stop driver
     *
     * After this function, the driver's thread will stop its
execution. The driver
     * and related variables should be properly setup. This function is
     * automatically called by IriBaseDriver::doStop(), an state
transition is
     * performed if return value equals true.
     *
     * \return bool successful
     */
     bool stopDriver(void);

    /**
     * \brief config update
     *
     * In this function the driver parameters must be updated with the
input
     * config variable. Then the new configuration state will be stored
in the
     * Config attribute.
     *
     * \param new_cfg the new driver configuration state
     *
     * \param level level in which the update is taken place
     */
     void config_update(Config& new_cfg, uint32_t level=0);

     // here define all delta_haptic_device_driver interface methods to
retrieve and set
     // the driver parameters

    /**
     * \brief Destructor
     *
     * This destructor is called when the object is about to be
destroyed.
     *
     */
     ~DeltaHapticDeviceDriver(void);
```

```
    /* retrieves the position p (in metres) and the angle o (in
degrees) of the End Effector */
    int getPositionAndOrientation (double& px, double& py, double& pz,
double& oa, double& ob, double& oc);

    /* retrieves the estimated instantaneous velocity v (in m/s) */
    int getLinearVelocity (double& vx, double& vy, double& vz);

    /* sets force f (in N) and torque t (in Nm) to the end effector*/
    int setForceAndTorque (double fx, double fy, double fz, double tx,
double ty, double tz);

};

#endif
```

3.5. delta_haptic_device_driver_node.h

```
// Copyright (C) 2010-2011 Institut de Robotica i Informatica
Industrial, CSIC-UPC.
// Author
// All rights reserved.
//
// This file is part of iri-ros-pkg
// iri-ros-pkg is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
published by
// the Free Software Foundation, either version 3 of the License, or
// at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
License
// along with this program.  If not, see
<http://www.gnu.org/licenses/>.
//
// IMPORTANT NOTE: This code has been generated through a script from
the
// iri_ros_scripts. Please do NOT delete any comments to guarantee the
correctness
// of the scripts. ROS topics can be easly add by using those scripts.
Please
// refer to the IRI wiki page for more information:
// http://wikiri.upc.es/index.php/Robotics_Lab

#ifndef _delta_haptic_device_driver_node_h_
#define _delta_haptic_device_driver_node_h_

#include <iri_base_driver/iri_base_driver_node.h>
#include "delta_haptic_device_driver.h"

// [publisher subscriber headers]
#include <geometry_msgs/Pose.h>
#include <geometry_msgs/WrenchStamped.h>
#include <geometry_msgs/Vector3.h>
#include <geometry_msgs/PoseStamped.h>
#include <std_msgs/Float64.h>
#include <std_msgs/Bool.h>

// [service client headers]
#include <iri_delta_haptic_device/ForceAndTorque.h>
```

```
// [action server client headers]

/**
 * \brief IRI ROS Specific Driver Class
 *
 * This class inherits from the IRI Core class
IriBaseNodeDriver<IriBaseDriver>,
 * to provide an execution thread to the driver object. A complete
framework
 * with utilites to test the node functionallity or to add diagnostics
to
 * specific situations is also given. The inherit template design form
allows
 * complete access to any IriBaseDriver object implementation.
 *
 * As mentioned, tests in the different driver states can be performed
through
 * class methods such as addNodeOpenedTests() or
addNodeRunningTests(). Tests
 * common to all nodes may be also executed in the pattern class
IriBaseNodeDriver.
 * Similarly to the tests, diagnostics can easyly be added. See ROS
Wiki for
 * more details:
 * http://www.ros.org/wiki/diagnostics/ (Tutorials: Creating a
Diagnostic Analyzer)
 * http://www.ros.org/wiki/self_test/ (Example: Self Test)
 */
class DeltaHapticDeviceDriverNode : public
iri_base_driver::IriBaseNodeDriver<DeltaHapticDeviceDriver>
{
  double px, py, pz;
  double oa, ob, oc;
  double vx, vy, vz;
  double fx, fy, fz;
  double tx, ty, tz;
  double pxi, pyi, pzi;
  double oai, obi, oci;
  double dsf, asf;
  bool pospos, force_feedback, torque_feedback;

  private:
    // [publisher attributes]
    ros::Publisher Initial_Position_publisher_;
    geometry_msgs::Pose Initial_Position_Pose_msg_;

    ros::Publisher Velocity_publisher_;
    geometry_msgs::Vector3 Velocity_Vector3_msg_;
```

```
    ros::Publisher Position_publisher_;
    geometry_msgs::PoseStamped Position_PoseStamped_msg_;


    // [subscriber attributes]
    ros::Subscriber ft_data_subscriber_;
    void ft_data_callback(const
geometry_msgs::WrenchStamped::ConstPtr& msg);
    pthread_mutex_t ft_data_mutex_;
    void ft_data_mutex_enter(void);
    void ft_data_mutex_exit(void);

    ros::Subscriber dsf_subscriber_;
    void dsf_callback(const std_msgs::Float64::ConstPtr& msg);
    pthread_mutex_t dsf_mutex_;
    void dsf_mutex_enter(void);
    void dsf_mutex_exit(void);

    ros::Subscriber asf_subscriber_;
    void asf_callback(const std_msgs::Float64::ConstPtr& msg);
    pthread_mutex_t asf_mutex_;
    void asf_mutex_enter(void);
    void asf_mutex_exit(void);

    ros::Subscriber state_subscriber_;
    void state_callback(const std_msgs::Bool::ConstPtr& msg);
    pthread_mutex_t state_mutex_;
    void state_mutex_enter(void);
    void state_mutex_exit(void);

    ros::Subscriber force_feedback_subscriber_;
    void force_feedback_callback(const std_msgs::Bool::ConstPtr& msg);
    pthread_mutex_t force_feedback_mutex_;
    void force_feedback_mutex_enter(void);
    void force_feedback_mutex_exit(void);

    ros::Subscriber torque_feedback_subscriber_;
    void torque_feedback_callback(const std_msgs::Bool::ConstPtr&
msg);
    pthread_mutex_t torque_feedback_mutex_;
    void torque_feedback_mutex_enter(void);
    void torque_feedback_mutex_exit(void);



    // [service attributes]
    ros::ServiceServer setForceAndTorque_server_;
```

```
    bool
setForceAndTorqueCallback(iri_delta_haptic_device::ForceAndTorque::Req
uest &req, iri_delta_haptic_device::ForceAndTorque::Response &res);
    pthread_mutex_t setForceAndTorque_mutex_;
    void setForceAndTorque_mutex_enter(void);
    void setForceAndTorque_mutex_exit(void);


    // [client attributes]

    // [action server attributes]

    // [action client attributes]

   /**
    * \brief post open hook
    *
    * This function is called by IriBaseNodeDriver::postOpenHook(). In
this function
    * specific parameters from the driver must be added so the ROS
dynamic
    * reconfigure application can update them.
    */
    void postNodeOpenHook(void);

  public:
   /**
    * \brief constructor
    *
    * This constructor mainly creates and initializes the
DeltaHapticDeviceDriverNode topics
    * through the given public_node_handle object. IriBaseNodeDriver
attributes
    * may be also modified to suit node specifications.
    *
    * All kind of ROS topics (publishers, subscribers, servers or
clients) can
    * be easyly generated with the scripts in the iri_ros_scripts
package. Refer
    * to ROS and IRI Wiki pages for more details:
    *
    *
http://www.ros.org/wiki/ROS/Tutorials/WritingPublisherSubscriber(c++)
    * http://www.ros.org/wiki/ROS/Tutorials/WritingServiceClient(c++)
    * http://wikiri.upc.es/index.php/Robotics_Lab
    *
    * \param nh a reference to the node handle object to manage all
ROS topics.
    */
```

```
    DeltaHapticDeviceDriverNode(ros::NodeHandle& nh);

   /**
    * \brief Destructor
    *
    * This destructor is called when the object is about to be
destroyed.
    *
    */
    ~DeltaHapticDeviceDriverNode(void);

  protected:
   /**
    * \brief main node thread
    *
    * This is the main thread node function. Code written here will be
executed
    * in every node loop while the driver is on running state. Loop
frequency
    * can be tuned by modifying loop_rate attribute.
    *
    * Here data related to the process loop or to ROS topics (mainly
data structs
    * related to the MSG and SRV files) must be updated. ROS publisher
objects
    * must publish their data in this process. ROS client servers may
also
    * request data to the corresponding server topics.
    */
    void mainNodeThread(void);

    // [diagnostic functions]

   /**
    * \brief node add diagnostics
    *
    * In this function ROS diagnostics applied to this specific node
may be
    * added. Common use diagnostics for all nodes are already called
from
    * IriBaseNodeDriver::addDiagnostics(), which also calls this
function. Information
    * of how ROS diagnostics work can be readen here:
    * http://www.ros.org/wiki/diagnostics/
    * http://www.ros.org/doc/api/diagnostic_updater/html/example_8cpp-
source.html
    */
    void addNodeDiagnostics(void);
```

```
    // [driver test functions]

   /**
    * \brief open status driver tests
    *
    * In this function tests checking driver's functionallity when
driver_base
    * status=open can be added. Common use tests for all nodes are
already called
    * from IriBaseNodeDriver tests methods. For more details on how
ROS tests work,
    * please refer to the Self Test example in:
    * http://www.ros.org/wiki/self_test/
    */
    void addNodeOpenedTests(void);

   /**
    * \brief stop status driver tests
    *
    * In this function tests checking driver's functionallity when
driver_base
    * status=stop can be added. Common use tests for all nodes are
already called
    * from IriBaseNodeDriver tests methods. For more details on how
ROS tests work,
    * please refer to the Self Test example in:
    * http://www.ros.org/wiki/self_test/
    */
    void addNodeStoppedTests(void);

   /**
    * \brief run status driver tests
    *
    * In this function tests checking driver's functionallity when
driver_base
    * status=run can be added. Common use tests for all nodes are
already called
    * from IriBaseNodeDriver tests methods. For more details on how
ROS tests work,
    * please refer to the Self Test example in:
    * http://www.ros.org/wiki/self_test/
    */
    void addNodeRunningTests(void);

   /**
    * \brief specific node dynamic reconfigure
    *
    * This function is called reconfigureHook()
    *
```

```
    * \param level integer
    */
    void reconfigureNodeHook(int level);

};

#endif
```

3.6. delta_haptic_device_driver_node.cpp

```
#include "delta_haptic_device_driver_node.h"
#include <math.h>
#include <unistd.h>

const double pi = 3.14159265;


DeltaHapticDeviceDriverNode::DeltaHapticDeviceDriverNode(ros::NodeHandle &nh)
:
  iri_base_driver::IriBaseNodeDriver<DeltaHapticDeviceDriver>(nh)
{
  //init class attributes if necessary
  this->loop_rate_ = 50;//in [Hz]
  driver_.setForceAndTorque(0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
  usleep(2500000);
  driver_.getPositionAndOrientation(this->pxi, this->pyi, this->pzi, this-
>oai, this->obi, this->oci);
  this->fx = this->fy = this->fz = this->tx = this->ty = this->tz = 0.0;
  this->dsf = this->asf = 1.0;
  this->pospos = false;
  this->force_feedback = this->torque_feedback = true;


  // [init publishers]
  this->Initial_Position_publisher_ = this-
>public_node_handle_.advertise<geometry_msgs::Pose>("Initial_Position", 10);
  this->Velocity_publisher_ = this-
>public_node_handle_.advertise<geometry_msgs::Vector3>("Velocity", 10);
  this->Position_publisher_ = this-
>public_node_handle_.advertise<geometry_msgs::PoseStamped>("Position", 10);

  // [init subscribers]
  this->ft_data_subscriber_ = this-
>public_node_handle_.subscribe("/ftc_force_sensor_driver_node/ft_data", 10,
&DeltaHapticDeviceDriverNode::ft_data_callback, this);
  pthread_mutex_init(&this->ft_data_mutex_,NULL);

  this->dsf_subscriber_ = this-
>public_node_handle_.subscribe("/rqt_telecommunication/dsf", 10,
&DeltaHapticDeviceDriverNode::dsf_callback, this);
  pthread_mutex_init(&this->dsf_mutex_,NULL);

  this->asf_subscriber_ = this-
>public_node_handle_.subscribe("/rqt_telecommunication/asf", 10,
&DeltaHapticDeviceDriverNode::asf_callback, this);
  pthread_mutex_init(&this->asf_mutex_,NULL);

  this->state_subscriber_ = this-
>public_node_handle_.subscribe("/rqt_telecommunication/dhd_state", 10,
&DeltaHapticDeviceDriverNode::state_callback, this);
  pthread_mutex_init(&this->state_mutex_,NULL);
```

```
  this->force_feedback_subscriber_ = this-
>public_node_handle_.subscribe("/rqt_telecommunication/dhd_force_feedback",
10, &DeltaHapticDeviceDriverNode::force_feedback_callback, this);
  pthread_mutex_init(&this->force_feedback_mutex_,NULL);

  this->torque_feedback_subscriber_ = this-
>public_node_handle_.subscribe("/rqt_telecommunication/dhd_torque_feedback",
10, &DeltaHapticDeviceDriverNode::torque_feedback_callback, this);
  pthread_mutex_init(&this->torque_feedback_mutex_,NULL);


  // [init services]
  this->setForceAndTorque_server_ = this-
>public_node_handle_.advertiseService("setForceAndTorque",
&DeltaHapticDeviceDriverNode::setForceAndTorqueCallback, this);
  pthread_mutex_init(&this->setForceAndTorque_mutex_,NULL);


  // [init clients]

  // [init action servers]

  // [init action clients]
}

void DeltaHapticDeviceDriverNode::mainNodeThread(void)
{


  //lock access to driver if necessary
  this->driver_.lock();
  if (this->pospos)
    driver_.setForceAndTorque(0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
  else
    if (this->force_feedback && this->torque_feedback)
      driver_.setForceAndTorque(this->fx, this->fy, this->fz, this->tx, this-
>ty, this->tz);
    else if (this->force_feedback && !(this->torque_feedback))
      driver_.setForceAndTorque(this->fx, this->fy, this->fz, 0.0, 0.0, 0.0);
    else if (!(this->force_feedback) && this->torque_feedback)
      driver_.setForceAndTorque(0.0, 0.0, 0.0, this->tx, this->ty, this->tz);
    else
      driver_.setForceAndTorque(0.0, 0.0, 0.0, 0.0, 0.0, 0.0);

  driver_.getPositionAndOrientation(this->px, this->py, this->pz, this->oa,
this->ob, this->oc);
  driver_.getLinearVelocity(this->vx, this->vy, this->vz);


  // [fill msg Header if necessary]

  // [fill msg structures]
  // Initialize the topic message structure
  this->Initial_Position_Pose_msg_.position.x = this->pxi;
  this->Initial_Position_Pose_msg_.position.y = this->pyi;
```

```
  this->Initial_Position_Pose_msg_.position.z = this->pzi;
  this->Initial_Position_Pose_msg_.orientation.x = sin(this-
>oai*pi/360)*cos(this->obi*pi/360)*cos(this->oci*pi/360)-cos(this-
>oai*pi/360)*sin(this->obi*pi/360)*sin(this->oci*pi/360);
  this->Initial_Position_Pose_msg_.orientation.y = cos(this-
>oai*pi/360)*sin(this->obi*pi/360)*cos(this->oci*pi/360)+sin(this-
>oai*pi/360)*cos(this->obi*pi/360)*sin(this->oci*pi/360);
  this->Initial_Position_Pose_msg_.orientation.z = cos(this-
>oai*pi/360)*cos(this->obi*pi/360)*sin(this->oci*pi/360)-sin(this-
>oai*pi/360)*sin(this->obi*pi/360)*cos(this->oci*pi/360);
  this->Initial_Position_Pose_msg_.orientation.w = cos(this-
>oai*pi/360)*cos(this->obi*pi/360)*cos(this->oci*pi/360)+sin(this-
>oai*pi/360)*sin(this->obi*pi/360)*sin(this->oci*pi/360);

  // Initialize the topic message structure
  this->Velocity_Vector3_msg_.x = this->vx;
  this->Velocity_Vector3_msg_.y = this->vy;
  this->Velocity_Vector3_msg_.z = this->vz;


  // Initialize the topic message structure

  this->Position_PoseStamped_msg_.header.seq = 30;
  this->Position_PoseStamped_msg_.header.stamp = ros::Time::now();
  this->Position_PoseStamped_msg_.header.frame_id = "End_Effector";

  this->Position_PoseStamped_msg_.pose.position.x = this->dsf*(this->px-this-
>pxi);
  this->Position_PoseStamped_msg_.pose.position.y = this->dsf*(this->py-this-
>pyi);
  this->Position_PoseStamped_msg_.pose.position.z = this->dsf*(this->pz-this-
>pzi);
  this->Position_PoseStamped_msg_.pose.orientation.x = sin(this->asf*(this-
>oa-this->oai)*pi/360)*cos(this->asf*(this->ob-this->obi)*pi/360)*cos(this-
>asf*(this->oc-this->oci)*pi/360)-cos(this->asf*(this->oa-this-
>oai)*pi/360)*sin(this->asf*(this->ob-this->obi)*pi/360)*sin(this->asf*(this-
>oc-this->oci)*pi/360);
  this->Position_PoseStamped_msg_.pose.orientation.y = cos(this->asf*(this-
>oa-this->oai)*pi/360)*sin(this->asf*(this->ob-this->obi)*pi/360)*cos(this-
>asf*(this->oc-this->oci)*pi/360)+sin(this->asf*(this->oa-this-
>oai)*pi/360)*cos(this->asf*(this->ob-this->obi)*pi/360)*sin(this->asf*(this-
>oc-this->oci)*pi/360);
  this->Position_PoseStamped_msg_.pose.orientation.z = cos(this->asf*(this-
>oa-this->oai)*pi/360)*cos(this->asf*(this->ob-this->obi)*pi/360)*sin(this-
>asf*(this->oc-this->oci)*pi/360)-sin(this->asf*(this->oa-this-
>oai)*pi/360)*sin(this->asf*(this->ob-this->obi)*pi/360)*cos(this->asf*(this-
>oc-this->oci)*pi/360);
  this->Position_PoseStamped_msg_.pose.orientation.w = cos(this->asf*(this-
>oa-this->oai)*pi/360)*cos(this->asf*(this->ob-this->obi)*pi/360)*cos(this-
>asf*(this->oc-this->oci)*pi/360)+sin(this->asf*(this->oa-this-
>oai)*pi/360)*sin(this->asf*(this->ob-this->obi)*pi/360)*sin(this->asf*(this-
>oc-this->oci)*pi/360);

  // Initialize the topic message structure
```

```
  // [fill action structure and make request to the action server]

  // [publish messages]
  // Uncomment the following line to publish the topic message
  this->Initial_Position_publisher_.publish(this->Initial_Position_Pose_msg_);

  // Uncomment the following line to publish the topic message
  this->Velocity_publisher_.publish(this->Velocity_Vector3_msg_);

  // Uncomment the following line to publish the topic message
  this->Position_publisher_.publish(this->Position_PoseStamped_msg_);


  //unlock access to driver if previously blocked
  this->driver_.unlock();
}

/*  [subscriber callbacks] */
void DeltaHapticDeviceDriverNode::ft_data_callback(const
geometry_msgs::WrenchStamped::ConstPtr& msg)
{
  //ROS_INFO("DeltaHapticDeviceDriverNode::ft_data_callback: New Message
Received");

  //use appropiate mutex to shared variables if necessary
  //this->driver_.lock();
  //this->ft_data_mutex_enter();

  void DeltaHapticDeviceDriverNode::ft_data_callback(const
geometry_msgs::WrenchStamped::ConstPtr& msg)

{

  this->fx = msg->wrench.force.z;

  this->fy = -0.5*(msg->wrench.force.x)+0.866*(msg->wrench.force.y);

  this->fz = -0.866(msg->wrench.force.x)-0.5(msg->wrench.force.y);

  this->tx = msg->wrench.torque.z;

  this->ty = -0.5*(msg->wrench.torque.x)+0.866*(msg->wrench.torque.y);

  this->tz = -0.866(msg->wrench.torque.x)-0.5(msg->wrench.torque.y);

}



  //unlock previously blocked shared variables
  //this->driver_.unlock();
  //this->ft_data_mutex_exit();
}


void DeltaHapticDeviceDriverNode::ft_data_mutex_enter(void)
{
  pthread_mutex_lock(&this->ft_data_mutex_);
}
```

```
void DeltaHapticDeviceDriverNode::dsf_callback(const
std_msgs::Float64::ConstPtr& msg)
{
  this->dsf = msg->data;
}

void DeltaHapticDeviceDriverNode::dsf_mutex_enter(void)
{
  pthread_mutex_lock(&this->dsf_mutex_);
}

void DeltaHapticDeviceDriverNode::asf_callback(const
std_msgs::Float64::ConstPtr& msg)
{
  this->asf = msg->data;
}

void DeltaHapticDeviceDriverNode::asf_mutex_enter(void)
{
  pthread_mutex_lock(&this->asf_mutex_);
}

void DeltaHapticDeviceDriverNode::state_callback(const
std_msgs::Bool::ConstPtr& msg)
{
  this->pospos = msg->data;
}

void DeltaHapticDeviceDriverNode::state_mutex_enter(void)
{
  pthread_mutex_lock(&this->state_mutex_);
}

void DeltaHapticDeviceDriverNode::force_feedback_callback(const
std_msgs::Bool::ConstPtr& msg)
{
  this->force_feedback = msg->data;
}

void DeltaHapticDeviceDriverNode::force_feedback_mutex_enter(void)
{
  pthread_mutex_lock(&this->force_feedback_mutex_);
}

void DeltaHapticDeviceDriverNode::torque_feedback_callback(const
std_msgs::Bool::ConstPtr& msg)
{
  this->torque_feedback = msg->data;
}

void DeltaHapticDeviceDriverNode::torque_feedback_mutex_enter(void)
{
  pthread_mutex_lock(&this->torque_feedback_mutex_);
}
```

```
/*  [service callbacks] */
bool
DeltaHapticDeviceDriverNode::setForceAndTorqueCallback(iri_delta_haptic_device
::ForceAndTorque::Request &req,
iri_delta_haptic_device::ForceAndTorque::Response &res)
{
  ROS_INFO("DeltaHapticDeviceDriverNode::setForceAndTorqueCallback: New
Request Received!");

  //use appropiate mutex to shared variables if necessary
  //this->driver_.lock();
  //this->setForceAndTorque_mutex_enter();

  //ROS_INFO("DeltaHapticDeviceDriverNode::setForceAndTorqueCallback:
Processing New Request!");
  //do operations with req and output on res
  if (this->fx != req.new_fx || this->fy != req.new_fy || this->fz !=
req.new_fz || this->tx != req.new_tx || this->ty != req.new_ty || this->tz !=
req.new_tz)
  {
    this->fx = req.new_fx;
    this->fy = req.new_fy;
    this->fz = req.new_fz;
    this->tx = req.new_tx;
    this->ty = req.new_ty;
    this->tz = req.new_tz;

    res.success = true;
  }

  else
    res.success = false;

  //unlock previously blocked shared variables
  //this->setForceAndTorque_mutex_exit();
  //this->driver_.unlock();

  return true;
}

void DeltaHapticDeviceDriverNode::setForceAndTorque_mutex_enter(void)
{
  pthread_mutex_lock(&this->setForceAndTorque_mutex_);
}

void DeltaHapticDeviceDriverNode::setForceAndTorque_mutex_exit(void)
{
  pthread_mutex_unlock(&this->setForceAndTorque_mutex_);
}


/*  [action callbacks] */
```

```
/*  [action requests] */

/*  [free dynamic memory] */

void DeltaHapticDeviceDriverNode::postNodeOpenHook(void)
{
}

void DeltaHapticDeviceDriverNode::addNodeDiagnostics(void)
{
}

void DeltaHapticDeviceDriverNode::addNodeOpenedTests(void)
{
}

void DeltaHapticDeviceDriverNode::addNodeStoppedTests(void)
{
}

void DeltaHapticDeviceDriverNode::addNodeRunningTests(void)
{
}

void DeltaHapticDeviceDriverNode::reconfigureNodeHook(int level)
{
}

DeltaHapticDeviceDriverNode::~DeltaHapticDeviceDriverNode(void)
{
}

/* main function */
int main(int argc,char *argv[])
{
  return driver_base::main<DeltaHapticDeviceDriverNode>(argc, argv,
"delta_haptic_device_driver_node");
}
```

3.7. ForceAndTorque.srv

float64 new_fx

float64 new_fy

float64 new_fz

float64 new_tx

float64 new_ty

float64 new_tz

---

bool success

4. Annex D: Codi de l'aplicació gràfica

4.1. plugin.xml

```
<library path="lib/libiri_gui_delta_haptic_device">
  <class name="Delta_haptic_device"
type="iri_gui_delta_haptic_device::Delta_haptic_device"
base_class_type="rqt_gui_cpp::Plugin">
    <description>
      A C++ GUI plugin to display the parameters of the Delta Haptic
Device delta_haptic_device.
    </description>
    <qtgui>
      <group>
        <label>Topics</label>
        <icon type="theme">folder</icon>
        <statustip>Plugins related to topics.</statustip>
      </group>
      <label>Delta_haptic_device</label>
      <icon type="theme">applications-other</icon>
      <statustip>A C++ GUI plugin to display the parameters of the
Delta Haptic Device delta_haptic_device.</statustip>
    </qtgui>
  </class>
</library>
```

4.2. delta_haptic_device.h

```
#ifndef iri_gui_delta_haptic_device_H
#define iri_gui_delta_haptic_device_H

#include <rqt_gui_cpp/plugin.h>

#include <ui_delta_haptic_device.h>

#include <ros/ros.h>
#include <rosbag/bag.h>
#include "ros/topic_manager.h"

#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/WrenchStamped.h>
#include <geometry_msgs/Vector3.h>
#include <sensor_msgs/JointState.h>
#include <std_msgs/Float64.h>
#include <std_msgs/Bool.h>
#include <rosgraph_msgs/Log.h>

#include <QWidget>
#include <QMutex>

#include <iri_delta_haptic_device/ForceAndTorque.h>
#include <std_srvs/Empty.h>
#include <iri_common_drivers_msgs/QueryJointsMovement.h>
#include <iri_common_drivers_msgs/QueryCartesianMovement.h>


namespace iri_gui_delta_haptic_device {

class Delta_haptic_device
  : public rqt_gui_cpp::Plugin
{
  Q_OBJECT
  double px, py, pz;
  double pxi, pyi, pzi;
  double oa, ob, oc;
  double oai, obi, oci;
  double vx, vy, vz;
  double fx, fy, fz;
  double tx, ty, tz;
  double q1, q2, q3, q4, q5, q6;
  double sr_x, sr_y, sr_z, sr_qx, sr_qy, sr_qz;
  double x0, y0, z0, qx0, qy0, qz0;
  double xT, yT, zT, qxT, qyT, qzT;
  double man_nom_vel;
  double progressBar;
```

```
  QString LogMessage, type, colour;
  bool store, show;
  int count;

public:
  Delta_haptic_device();
  virtual void initPlugin(qt_gui_cpp::PluginContext& context);
  virtual void shutdownPlugin();



protected:
  virtual void dhd_position_callback(const
geometry_msgs::PoseStamped::ConstPtr& msg);
  virtual void dhd_initial_position_callback(const
geometry_msgs::Pose::ConstPtr& msg);
  virtual void dhd_velocity_callback(const
geometry_msgs::Vector3::ConstPtr& msg);
  virtual void ft_data_callback(const
geometry_msgs::WrenchStamped::ConstPtr& msg);
  virtual void staubli_joints_callback(const
sensor_msgs::JointState::ConstPtr& msg);
  virtual void staubli_tcp_callback(const
geometry_msgs::PoseStamped::ConstPtr& msg);
  virtual void log_callback(const rosgraph_msgs::Log::ConstPtr& msg);

signals:
  void PositionChanged();
  void initialPosition();
  void VelocityChanged();
  void Ft_dataChanged();
  void JointsChanged();
  void TCPChanged();
  void DebugChanged(QString msg);
  void InfoChanged(QString msg);
  void WarnChanged(QString msg);
  void ErrorChanged(QString msg);
  void FatalChanged(QString msg);
  void storing(bool storing);


protected slots:
  virtual void onVelSliderManChanged(int value);
  virtual void PositionUpdate();
  virtual void initialPositionUpdate();
  virtual void VelocityUpdate();
  virtual void Ft_dataUpdate();
  virtual void JointsUpdate();
  virtual void TCPUpdate();
```

```
  virtual void setForceAndTorque();
  virtual void setZero();
  virtual void setDSF();
  virtual void setASF();
  virtual void setJoints();
  virtual void setCart();
  virtual void setNewOrigin();
  virtual void setNewTool();
  virtual void setState(bool state);
  virtual void setForceFeedback(bool force_feedback);
  virtual void setTorqueFeedback(bool torque_feedback);
  virtual void storeData();
  virtual void LogUpdate(QString msg);
  virtual void stopDataStoring();
  virtual void EqualJoints();
  virtual void EqualCart();



private:

  Ui::Delta_haptic_device ui_;

  QWidget* widget_;
  QMutex mutex_;

  ros::Subscriber dhd_position_subscriber_;
  geometry_msgs::PoseStamped dhd_position_msg_;
  ros::Subscriber dhd_initial_position_subscriber_;
  ros::Subscriber dhd_velocity_subscriber_;
  geometry_msgs::Vector3 dhd_velocity_msg_;
  ros::Subscriber ft_data_subscriber_;
  geometry_msgs::WrenchStamped ft_data_msg_;
  ros::Subscriber staubli_tcp_subscriber_;
  geometry_msgs::PoseStamped staubli_tcp_msg_;
  ros::Subscriber staubli_joints_subscriber_;
  sensor_msgs::JointState staubli_joints_msg_;
  ros::Subscriber log_subscriber_;

  ros::Publisher dsf_publisher_;
  std_msgs::Float64 dsf_Float64_msg_;
  ros::Publisher asf_publisher_;
  std_msgs::Float64 asf_Float64_msg_;
  ros::Publisher staubli_tcp_publisher_;
  geometry_msgs::PoseStamped staubli_tcp_PoseStamped_msg_;
  ros::Publisher staubli_joints_publisher_;
  sensor_msgs::JointState staubli_joints_JointState_msg_;
  ros::Publisher state_publisher_;
```

```
  std_msgs::Bool state_Bool_msg_;
  ros::Publisher force_feedback_publisher_;
  std_msgs::Bool force_feedback_Bool_msg_;
  ros::Publisher torque_feedback_publisher_;
  std_msgs::Bool torque_feedback_Bool_msg_;

  ros::ServiceClient setForceAndTorque_client_;
  iri_delta_haptic_device::ForceAndTorque setForceAndTorque_srv_;
  ros::ServiceClient setZero_client_;
  std_srvs::Empty setZero_srv_;
  ros::ServiceClient move_in_joints_client_;
  iri_common_drivers_msgs::QueryJointsMovement move_in_joints_srv_;
  ros::ServiceClient move_in_cart_client_;
  iri_common_drivers_msgs::QueryCartesianMovement move_in_cart_srv_;

  rosbag::Bag bag;
};


} // namespace
#endif // iri_gui_delta_haptic_device_H
```

4.3. delta_haptic_device.cpp

```
#include "delta_haptic_device.h"

#include <pluginlib/class_list_macros.h>

#include <ros/master.h>

#include <QStringList>

#include <math.h>
#include <vector>

const double pi = 3.14159265;

namespace iri_gui_delta_haptic_device {


Delta_haptic_device::Delta_haptic_device()
  : rqt_gui_cpp::Plugin()
  , widget_(0)
{
  setObjectName("Delta_haptic_device");
}

void Delta_haptic_device::initPlugin(qt_gui_cpp::PluginContext& context)
{
  // access standalone command line arguments
  QStringList argv = context.argv();
  // create QWidget
  widget_ = new QWidget();
  // extend the widget with all attributes and children from UI file
  ui_.setupUi(widget_);
  // add widget to the user interface
  context.addWidget(widget_);


/*****************************************************************************
*********/
//Init Class Atributes
/*****************************************************************************
*********/

  this->px = this->py = this->pz = 0.0;
  this->pxi = this->pyi = this->pzi = 0.0;
  this->oa = this->ob = this->oc = 0.0;
  this->oai = this->obi = this->oci = 0.0;
  this->vx = this->vy = this->vz = 0.0;
  this->fx = this->fy = this->fz = 0.0;
  this->tx = this->ty = this->tz = 0.0;
  this->q1 = this->q2 = this->q3 = this->q4 = this->q5 = this->q6 = 0.0;
  this->sr_x = this->sr_y = this->sr_z = this->sr_qx = this->sr_qy = this-
>sr_qz = 0.0;
  this->x0 = this->y0 = this->z0 = this->qx0 = this->qy0 = this->qz0 = 0.0;
```

```
  this->xT = this->yT = this->zT = this->qxT = this->qyT = this->qzT = 0.0;
  this->man_nom_vel = ui_.VelSliderMan->value();
  progressBar = 0.0;
  store = false;
  count = 1;



/*************************************************************************
*********/
//Init Subscribers
/*************************************************************************
*********/

  /* Subscribes to DHD Position */
  dhd_position_subscriber_ =
getNodeHandle().subscribe("/delta_haptic_device_driver_node/Position", 10,
&Delta_haptic_device::dhd_position_callback, this);

  /* Subscribes to DHD Initial Position */
  dhd_initial_position_subscriber_ =
getNodeHandle().subscribe("/delta_haptic_device_driver_node/Initial_Position",
10, &Delta_haptic_device::dhd_initial_position_callback, this);

  /* Subscribes to DHD Velocity */
  dhd_velocity_subscriber_ =
getNodeHandle().subscribe("/delta_haptic_device_driver_node/Velocity", 10,
&Delta_haptic_device::dhd_velocity_callback, this);

  /* Subscribes to ft_data */
  ft_data_subscriber_ =
getNodeHandle().subscribe("/ftc_force_sensor_driver_node/ft_data", 10,
&Delta_haptic_device::ft_data_callback, this);

  /* Subscribes to Staubli Joints */
  staubli_joints_subscriber_ = getNodeHandle().subscribe("/joint_states", 10,
&Delta_haptic_device::staubli_joints_callback, this);

  /* Subscribes to Staubli Cartesian */
  staubli_tcp_subscriber_ =
getNodeHandle().subscribe("/iri_staubli_controller/tcp_pose", 10,
&Delta_haptic_device::staubli_tcp_callback, this);

  /* Subscribes to rosout_agg */
  log_subscriber_ = getNodeHandle().subscribe("/rosout_agg", 10,
&Delta_haptic_device::log_callback, this);



/*************************************************************************
*************/
//Init Publishers
/*************************************************************************
*************/
```

```
  /* Publishes Distance Scale Factor */
  this->dsf_publisher_ = this->getNodeHandle().advertise<std_msgs::Float64>("/iri_gui_delta_haptic_device/dsf",10);

  /* Publishes Angle Scale Factor */
  this->asf_publisher_ = this->getNodeHandle().advertise<std_msgs::Float64>("/iri_gui_delta_haptic_device/asf",10);

  /* Publishes boolean to allow the force and torque feedback of the DHD (true = don't allow, false = allow) */
  this->state_publisher_ = this->getNodeHandle().advertise<std_msgs::Bool>("/iri_gui_delta_haptic_device/dhd_state",10);

  /* Publishes boolean to allow the force feedback of the DHD (true = allow, false = don't allow) */
  this->force_feedback_publisher_ = this->getNodeHandle().advertise<std_msgs::Bool>("/iri_gui_delta_haptic_device/dhd_force_feedback",10);

  /* Publishes boolean to allow the torque feedback of the DHD (true = allow, false = don't allow) */
  this->torque_feedback_publisher_ = this->getNodeHandle().advertise<std_msgs::Bool>("/iri_gui_delta_haptic_device/dhd_torque_feedback",10);

/*****************************************************************************/
//Init clients
/*****************************************************************************/

  setForceAndTorque_client_ = this->getNodeHandle().serviceClient<iri_delta_haptic_device::ForceAndTorque>("/delta_haptic_device_driver_node/setForceAndTorque");

  setZero_client_ = this->getNodeHandle().serviceClient<std_srvs::Empty>("/ftc_force_sensor_driver_node/set_zero");

  move_in_joints_client_ = this->getNodeHandle().serviceClient<iri_common_drivers_msgs::QueryJointsMovement>("/iri_staubli_controller/move_in_joints");

  move_in_cart_client_ = this->getNodeHandle().serviceClient<iri_common_drivers_msgs::QueryCartesianMovement>("/iri_staubli_controller/move_in_cart");

/*****************************************************************************/
//Arrange the connections
/*****************************************************************************/
```

```
  /* Changes the man_nom_vel variable and the browser text */
  connect(ui_.VelSliderMan, SIGNAL(valueChanged(int)), this,
SLOT(onVelSliderManChanged(int)));



  /* Allows the display of the DHD Position data */
  connect(this, SIGNAL(PositionChanged()), this, SLOT(PositionUpdate()),
Qt::BlockingQueuedConnection);

  /* Allows the display of the DHD Initial Position data */
  connect(this, SIGNAL(initialPosition()), this,
SLOT(initialPositionUpdate()), Qt::BlockingQueuedConnection);

  /* Allows the display of the DHD Velocity data */
  connect(this, SIGNAL(VelocityChanged()), this, SLOT(VelocityUpdate()),
Qt::BlockingQueuedConnection);

  /* Allows the display of the Ft_data data */
  connect(this, SIGNAL(Ft_dataChanged()), this, SLOT(Ft_dataUpdate()),
Qt::BlockingQueuedConnection);

  /* Allows the display of the Staubli Joints data */
  connect(this, SIGNAL(JointsChanged()), this, SLOT(JointsUpdate()),
Qt::BlockingQueuedConnection);

  /* Allows the display of the Staubli Cartesian data */
  connect(this, SIGNAL(TCPChanged()), this, SLOT(TCPUpdate()),
Qt::BlockingQueuedConnection);

  connect(this, SIGNAL(DebugChanged(QString)), this, SLOT(LogUpdate(QString)),
Qt::BlockingQueuedConnection);

  connect(this, SIGNAL(InfoChanged(QString)), this, SLOT(LogUpdate(QString)),
Qt::BlockingQueuedConnection);

  connect(this, SIGNAL(WarnChanged(QString)), this, SLOT(LogUpdate(QString)),
Qt::BlockingQueuedConnection);

  connect(this, SIGNAL(ErrorChanged(QString)), this, SLOT(LogUpdate(QString)),
Qt::BlockingQueuedConnection);

  connect(this, SIGNAL(FatalChanged(QString)), this, SLOT(LogUpdate(QString)),
Qt::BlockingQueuedConnection);

  connect(ui_.ClearBtt, SIGNAL(clicked()), ui_.LogArea, SLOT(clear()));

  connect(this, SIGNAL(storing(bool)), ui_.StopDataStoringBttTel,
SLOT(setEnabled(bool)));

  /* Starts storing data of the DHD, FTC Force Sensor and Staubli Robot */
  connect(ui_.StartDataStoringBttTel, SIGNAL(clicked()), this,
SLOT(storeData()));
```

```
  connect(ui_.StopDataStoringBttTel, SIGNAL(clicked()), this,
SLOT(stopDataStoring()));
```

```
  /* Conveys the input force and torque to the DHD */
  connect(ui_.SendForTorBtt, SIGNAL(clicked()), this,
SLOT(setForceAndTorque()));

  /* Sets the current force and torque as zero */
  connect(ui_.SetZeroButton, SIGNAL(clicked()), this, SLOT(setZero()));

  /* Moves the Staubli Robot in the joints space */
  connect(ui_.MoveJointsBtt, SIGNAL(clicked()), this, SLOT(setJoints()));

  /* Moves the Staubli Robot in the cartesian space */
  connect(ui_.MoveWorldBtt, SIGNAL(clicked()), this, SLOT(setCart()));
```

```
  /* Conveys the Distance Scale Factor */
  connect(ui_.DSF, SIGNAL(valueChanged(double)), this, SLOT(setDSF()));

  /* Conveys the Angle Scale Factor */
  connect(ui_.ASF, SIGNAL(valueChanged(double)), this, SLOT(setASF()));

  /* Activates the PosPos Mode (Neither Force nor Torque Feedback) */
  connect(ui_.PosPosButton, SIGNAL(toggled(bool)), this,
SLOT(setState(bool)));

  /* Activates the Force Mode (Force Feedback allowed) */
  connect(ui_.ForBox, SIGNAL(toggled(bool)), this,
SLOT(setForceFeedback(bool)));

  /* Activates the Torque Mode (Torque Feedback allowed) */
  connect(ui_.TorBox, SIGNAL(toggled(bool)), this,
SLOT(setTorqueFeedback(bool)));

  /* Sets a new origin for the Staubli Robot */
  connect(ui_.SetOriginBtt, SIGNAL(clicked()), this, SLOT(setNewOrigin()));

  /* Sets a new TCP */
  connect(ui_.SetToolBtt, SIGNAL(clicked()), this, SLOT(setNewTool()));

  /* Sets the current value of the joints to the joints spinbox */
  connect(ui_.EqualJoints, SIGNAL(clicked()), this, SLOT(EqualJoints()));

  /* Sets the current value of the cartesian space to the joints spinbox */
  connect(ui_.EqualCart, SIGNAL(clicked()), this, SLOT(EqualCart()));
```

```
/*****************************************************************************
***************/
//Set up the widgets
/*****************************************************************************
***************/

  ui_.mdhd_px->setText(QString::number(this->px));     //Displays the position
of the DHD
  ui_.mdhd_py->setText(QString::number(this->py));
  ui_.mdhd_pz->setText(QString::number(this->pz));
  ui_.tdhd_px->setText(QString::number(this->px));
  ui_.tdhd_py->setText(QString::number(this->py));
  ui_.tdhd_pz->setText(QString::number(this->pz));

  ui_.mdhd_pxi->setText(QString::number(this->pxi));     //Displays the
initial position of the DHD
  ui_.mdhd_pyi->setText(QString::number(this->pyi));
  ui_.mdhd_pzi->setText(QString::number(this->pzi));


  ui_.mdhd_oa->setText(QString::number(this->oa));     //Displays the
orientation of the DHD
  ui_.mdhd_ob->setText(QString::number(this->ob));
  ui_.mdhd_oc->setText(QString::number(this->oc));
  ui_.tdhd_oa->setText(QString::number(this->oa));
  ui_.tdhd_ob->setText(QString::number(this->ob));
  ui_.tdhd_oc->setText(QString::number(this->oc));

  ui_.mdhd_oai->setText(QString::number(this->oai));     //Displays the
orientation of the DHD
  ui_.mdhd_obi->setText(QString::number(this->obi));
  ui_.mdhd_oci->setText(QString::number(this->oci));

  ui_.mdhd_vx->setText(QString::number(this->vx));     //Displays the velocity
of the DHD
  ui_.mdhd_vy->setText(QString::number(this->vy));
  ui_.mdhd_vz->setText(QString::number(this->vz));
  ui_.tdhd_vx->setText(QString::number(this->vx));
  ui_.tdhd_vy->setText(QString::number(this->vy));
  ui_.tdhd_vz->setText(QString::number(this->vz));

  ui_.tfs_fx->setText(QString::number(this->fx));      //Displays the force
applied to the DHD
  ui_.tfs_fy->setText(QString::number(this->fy));
  ui_.tfs_fz->setText(QString::number(this->fz));

  ui_.tfs_tx->setText(QString::number(this->tx));      //Displays the torque
applied to the DHD
  ui_.tfs_ty->setText(QString::number(this->ty));
  ui_.tfs_tz->setText(QString::number(this->tz));
```

```
  ui_.msr_q1->setText(QString::number(this->q1));      //Displays the joint
values of the Staubli Robot
  ui_.msr_q2->setText(QString::number(this->q2));
  ui_.msr_q3->setText(QString::number(this->q3));
  ui_.msr_q4->setText(QString::number(this->q4));
  ui_.msr_q5->setText(QString::number(this->q5));
  ui_.msr_q6->setText(QString::number(this->q6));
  ui_.tsr_q1->setText(QString::number(this->q1));
  ui_.tsr_q2->setText(QString::number(this->q2));
  ui_.tsr_q3->setText(QString::number(this->q3));
  ui_.tsr_q4->setText(QString::number(this->q4));
  ui_.tsr_q5->setText(QString::number(this->q5));
  ui_.tsr_q6->setText(QString::number(this->q6));

  ui_.msr_x->setText(QString::number(this->sr_x));     //Displays the position
and orientation of the Staubli Robot
  ui_.msr_y->setText(QString::number(this->sr_y));
  ui_.msr_z->setText(QString::number(this->sr_z));
  ui_.msr_qx->setText(QString::number(this->sr_qx));
  ui_.msr_qy->setText(QString::number(this->sr_qy));
  ui_.msr_qz->setText(QString::number(this->sr_qz));
  ui_.tsr_x->setText(QString::number(this->sr_x));
  ui_.tsr_y->setText(QString::number(this->sr_y));
  ui_.tsr_z->setText(QString::number(this->sr_z));
  ui_.tsr_qx->setText(QString::number(this->sr_qx));
  ui_.tsr_qy->setText(QString::number(this->sr_qy));
  ui_.tsr_qz->setText(QString::number(this->sr_qz));

  ui_.msr_x0->setText(QString::number(this->x0));     //Displays the current
origin
  ui_.msr_y0->setText(QString::number(this->y0));
  ui_.msr_z0->setText(QString::number(this->z0));
  ui_.msr_qx0->setText(QString::number(this->qx0));
  ui_.msr_qy0->setText(QString::number(this->qy0));
  ui_.msr_qz0->setText(QString::number(this->qz0));

  ui_.NomVelBrowserMan->setText(QString::number(man_nom_vel));
//Displays the value of the current nominal velocity

  ui_.FileName->setText("test");
}



void Delta_haptic_device::shutdownPlugin()
{

//ros::TopicManager::unregisterPublisher("/iri_gui_delta_haptic_device/dsf");
  /*dhd_position_subscriber_.shutdown();
  dhd_initial_position_subscriber_.shutdown();
  dhd_velocity_subscriber_.shutdown();
  ft_data_subscriber_.shutdown();
  staubli_joints_subscriber_.shutdown();
  staubli_tcp_subscriber_.shutdown();*/
  ros::shutdown();
```

```
}


void Delta_haptic_device::dhd_position_callback(const
geometry_msgs::PoseStamped::ConstPtr& msg)
{
  this->px = msg->pose.position.x;
  this->py = msg->pose.position.y;
  this->pz = msg->pose.position.z;

  this->oa = 180/pi*atan2((2*(msg->pose.orientation.w*msg-
>pose.orientation.x+msg->pose.orientation.y*msg->pose.orientation.z)),(1-
2*(msg->pose.orientation.x*msg->pose.orientation.x+msg-
>pose.orientation.y*msg->pose.orientation.y))); //From quaternion to Euler
  this->ob = 180/pi*asin(2*(msg->pose.orientation.w*msg->pose.orientation.y-
msg->pose.orientation.z*msg->pose.orientation.x));
  this->oc = 180/pi*atan2((2*(msg->pose.orientation.w*msg-
>pose.orientation.z+msg->pose.orientation.x*msg->pose.orientation.y)),(1-
2*(msg->pose.orientation.y*msg->pose.orientation.y+msg-
>pose.orientation.z*msg->pose.orientation.z)));

  this->dhd_position_msg_.header.stamp = ros::Time::now();
  this->dhd_position_msg_.header.frame_id = "End_Effector";

  this->dhd_position_msg_.pose.position.x = 0;
  this->dhd_position_msg_.pose.position.y = 0;
  this->dhd_position_msg_.pose.position.z = 0;

  this->dhd_position_msg_.pose.orientation.x = 0;
  this->dhd_position_msg_.pose.orientation.y = 0;
  this->dhd_position_msg_.pose.orientation.z = 0;
  this->dhd_position_msg_.pose.orientation.w = 0;


  if (store)
  {
    if (ui_.store_position->isChecked())
    {
      this->dhd_position_msg_.pose.position.x = msg->pose.position.x;
      this->dhd_position_msg_.pose.position.y = msg->pose.position.y;
      this->dhd_position_msg_.pose.position.z = msg->pose.position.z;
    }
    if (ui_.store_orientation->isChecked())
    {
      this->dhd_position_msg_.pose.orientation.x = msg->pose.orientation.x;
      this->dhd_position_msg_.pose.orientation.y = msg->pose.orientation.y;
      this->dhd_position_msg_.pose.orientation.z = msg->pose.orientation.z;
      this->dhd_position_msg_.pose.orientation.w = msg->pose.orientation.w;
    }
    bag.write("/delta_haptic_device_driver_node/Position", ros::Time::now(),
this->dhd_position_msg_);
  }
  emit PositionChanged();
}
```

```
void Delta_haptic_device::dhd_initial_position_callback(const
geometry_msgs::Pose::ConstPtr& msg)
{
  this->pxi = msg->position.x;
  this->pyi = msg->position.y;
  this->pzi = msg->position.z;
  this->oai = 180/pi*atan2((2*(msg->orientation.w*msg->orientation.x+msg-
>orientation.y*msg->orientation.z)),(1-2*(msg->orientation.x*msg-
>orientation.x+msg->orientation.y*msg->orientation.y))); //From quaternion to
Euler
  this->obi = 180/pi*asin(2*(msg->orientation.w*msg->orientation.y-msg-
>orientation.z*msg->orientation.x));
  this->oci = 180/pi*atan2((2*(msg->orientation.w*msg->orientation.z+msg-
>orientation.x*msg->orientation.y)),(1-2*(msg->orientation.y*msg-
>orientation.y+msg->orientation.z*msg->orientation.z)));



  emit initialPosition();

}

void Delta_haptic_device::dhd_velocity_callback(const
geometry_msgs::Vector3::ConstPtr& msg)
{
  this->vx = msg->x;
  this->vy = msg->y;
  this->vz = msg->z;

  dhd_velocity_msg_.x = msg->x;
  dhd_velocity_msg_.y = msg->y;
  dhd_velocity_msg_.z = msg->z;

  if (store && ui_.store_velocity->isChecked())
    bag.write("/delta_haptic_device_driver_node/Velocity", ros::Time::now(),
dhd_velocity_msg_);

  emit VelocityChanged();
}

void Delta_haptic_device::ft_data_callback(const
geometry_msgs::WrenchStamped::ConstPtr& msg)
{
  this->fx = msg->wrench.force.x;
  this->fy = msg->wrench.force.y;
  this->fz = msg->wrench.force.z;
  this->tx = msg->wrench.torque.x;
  this->ty = msg->wrench.torque.y;
  this->tz = msg->wrench.torque.z;

  ft_data_msg_.wrench.force.x = 0;
  ft_data_msg_.wrench.force.y = 0;
  ft_data_msg_.wrench.force.z = 0;
  ft_data_msg_.wrench.torque.x = 0;
  ft_data_msg_.wrench.torque.y = 0;
```

```
  ft_data_msg_.wrench.torque.z = 0;

  if (store)
  {
    if (ui_.store_forces->isChecked())
    {
      ft_data_msg_.wrench.force.x = msg->wrench.force.x;
      ft_data_msg_.wrench.force.y = msg->wrench.force.y;
      ft_data_msg_.wrench.force.z = msg->wrench.force.z;
    }
    if (ui_.store_torques->isChecked())
    {
      ft_data_msg_.wrench.torque.x = msg->wrench.torque.x;
      ft_data_msg_.wrench.torque.y = msg->wrench.torque.y;
      ft_data_msg_.wrench.torque.z = msg->wrench.torque.z;
    }
    bag.write("/ftc_force_sensor/ft_data", ros::Time::now(), ft_data_msg_);
  }
  emit Ft_dataChanged();
}

void Delta_haptic_device::staubli_joints_callback(const
sensor_msgs::JointState::ConstPtr& msg)
{
  q1 = msg->position[0]*180/pi;
  q2 = msg->position[1]*180/pi;
  q3 = msg->position[2]*180/pi;
  q4 = msg->position[3]*180/pi;
  q5 = msg->position[4]*180/pi;
  q6 = msg->position[5]*180/pi;

  std::vector<double> angles(6);
  angles[0] = msg->position[0];
  angles[1] = msg->position[1];
  angles[2] = msg->position[2];
  angles[3] = msg->position[3];
  angles[4] = msg->position[4];
  angles[5] = msg->position[5];

  staubli_joints_msg_.position = angles;

  if (store && ui_.store_joints->isChecked())
    bag.write("/joint_states", ros::Time::now(), staubli_joints_msg_);

  emit JointsChanged();
}

void Delta_haptic_device::staubli_tcp_callback(const
geometry_msgs::PoseStamped::ConstPtr& msg)
{
  this->sr_x = msg->pose.position.x;
  this->sr_y = msg->pose.position.y;
  this->sr_z = msg->pose.position.z;
  this->sr_qx = 180/pi*atan2((2*(msg->pose.orientation.w*msg-
>pose.orientation.x+msg->pose.orientation.y*msg->pose.orientation.z)),(1-
```

```
2*(msg->pose.orientation.x*msg->pose.orientation.x+msg-
>pose.orientation.y*msg->pose.orientation.y)));
  this->sr_qy = 180/pi*asin(2*(msg->pose.orientation.w*msg-
>pose.orientation.y-msg->pose.orientation.z*msg->pose.orientation.x));
  this->sr_qz = 180/pi*atan2((2*(msg->pose.orientation.w*msg-
>pose.orientation.z+msg->pose.orientation.x*msg->pose.orientation.y)),(1-
2*(msg->pose.orientation.y*msg->pose.orientation.y+msg-
>pose.orientation.z*msg->pose.orientation.z)));

  staubli_tcp_msg_.pose.position.x = msg->pose.position.x;
  staubli_tcp_msg_.pose.position.y = msg->pose.position.y;
  staubli_tcp_msg_.pose.position.z = msg->pose.position.z;

  staubli_tcp_msg_.pose.orientation.x = msg->pose.orientation.x;
  staubli_tcp_msg_.pose.orientation.y = msg->pose.orientation.y;
  staubli_tcp_msg_.pose.orientation.z = msg->pose.orientation.z;
  staubli_tcp_msg_.pose.orientation.w = msg->pose.orientation.w;

  if (store && ui_.store_cartesian->isChecked())
    bag.write("/iri_staubli_controller/tcp_pose", ros::Time::now(),
staubli_tcp_msg_);

  emit TCPChanged();
}

void Delta_haptic_device::log_callback(const rosgraph_msgs::Log::ConstPtr&
msg)
{
  LogMessage = QString::fromStdString(msg->msg);
  if (msg->level == 1)
    {
    show = ui_.show_debug->isChecked();
    type = "[DEBUG]";
    colour = "black";
    emit DebugChanged(LogMessage);
    }
  else if (msg->level == 2)
    {
    show = ui_.show_info->isChecked();
    type = "[INFO]";
    colour = "black";
    emit InfoChanged(LogMessage);
    }
  else if (msg->level == 4)
    {
    show = ui_.show_warn->isChecked();
    type = "[WARN]";
    colour = "orange";
    emit WarnChanged(LogMessage);
    }
  else if (msg->level == 8)
    {
    show = ui_.show_error->isChecked();
    type = "[ERROR]";
    colour = "darkRed";
```

```
    emit ErrorChanged(LogMessage);
    }
  else if (msg->level ==16)
    {
    show = ui_.show_fatal->isChecked();
    type = "[FATAL]";
    colour = "red";
    emit FatalChanged(LogMessage);
    }
}

void Delta_haptic_device::setForceAndTorque()
{
// [fill srv structure and make request to the server]
  setForceAndTorque_srv_.request.new_fx = ui_.FxDsB->value();
  setForceAndTorque_srv_.request.new_fy = ui_.FyDsB->value();
  setForceAndTorque_srv_.request.new_fz = ui_.FzDsB->value();
  setForceAndTorque_srv_.request.new_tx = ui_.TxDsB->value();
  setForceAndTorque_srv_.request.new_ty = ui_.TyDsB->value();
  setForceAndTorque_srv_.request.new_tz = ui_.TzDsB->value();

  setForceAndTorque_client_.call(setForceAndTorque_srv_);
}

void Delta_haptic_device::setZero()
{
  setZero_client_.call(setZero_srv_);
}


void Delta_haptic_device::onVelSliderManChanged(int value)
{
  ui_.NomVelBrowserMan->setText(QString::number(value));
  this->man_nom_vel = value;
}

void Delta_haptic_device::PositionUpdate()
{
  double px_ = (this->px)/ui_.DSF->value();
  double py_ = (this->py)/ui_.DSF->value();
  double pz_ = (this->pz)/ui_.DSF->value();
  double oa_ = (this->oa)/ui_.ASF->value();
  double ob_ = (this->ob)/ui_.ASF->value();
  double oc_ = (this->oc)/ui_.ASF->value();

  ui_.mdhd_px->setText(QString::number(px_));
  ui_.mdhd_py->setText(QString::number(py_));
  ui_.mdhd_pz->setText(QString::number(pz_));
  ui_.tdhd_px->setText(QString::number(px_));
  ui_.tdhd_py->setText(QString::number(py_));
  ui_.tdhd_pz->setText(QString::number(pz_));

  ui_.mdhd_oa->setText(QString::number(oa_));
  ui_.mdhd_ob->setText(QString::number(ob_));
  ui_.mdhd_oc->setText(QString::number(oc_));
```

```
  ui_.tdhd_oa->setText(QString::number(oa_));
  ui_.tdhd_ob->setText(QString::number(ob_));
  ui_.tdhd_oc->setText(QString::number(oc_));
}

void Delta_haptic_device::initialPositionUpdate()
{
  ui_.mdhd_pxi->setText(QString::number(this->pxi));
  ui_.mdhd_pyi->setText(QString::number(this->pyi));
  ui_.mdhd_pzi->setText(QString::number(this->pzi));
  ui_.mdhd_oai->setText(QString::number(this->oai));
  ui_.mdhd_obi->setText(QString::number(this->obi));
  ui_.mdhd_oci->setText(QString::number(this->oci));
}

void Delta_haptic_device::VelocityUpdate()
{
  ui_.mdhd_vx->setText(QString::number(this->vx));
  ui_.mdhd_vy->setText(QString::number(this->vy));
  ui_.mdhd_vz->setText(QString::number(this->vz));
  ui_.tdhd_vx->setText(QString::number(this->vx));
  ui_.tdhd_vy->setText(QString::number(this->vy));
  ui_.tdhd_vz->setText(QString::number(this->vz));
}

void Delta_haptic_device::Ft_dataUpdate()
{
  ui_.tfs_fx->setText(QString::number(this->fx));
  ui_.tfs_fy->setText(QString::number(this->fy));
  ui_.tfs_fz->setText(QString::number(this->fz));

  ui_.tfs_tx->setText(QString::number(this->tx));
  ui_.tfs_ty->setText(QString::number(this->ty));
  ui_.tfs_tz->setText(QString::number(this->tz));
}

void Delta_haptic_device::JointsUpdate()
{
  ui_.msr_q1->setText(QString::number(this->q1));
  ui_.msr_q2->setText(QString::number(this->q2));
  ui_.msr_q3->setText(QString::number(this->q3));
  ui_.msr_q4->setText(QString::number(this->q4));
  ui_.msr_q5->setText(QString::number(this->q5));
  ui_.msr_q6->setText(QString::number(this->q6));
  ui_.tsr_q1->setText(QString::number(this->q1));
  ui_.tsr_q2->setText(QString::number(this->q2));
  ui_.tsr_q3->setText(QString::number(this->q3));
  ui_.tsr_q4->setText(QString::number(this->q4));
  ui_.tsr_q5->setText(QString::number(this->q5));
  ui_.tsr_q6->setText(QString::number(this->q6));
}

void Delta_haptic_device::TCPUpdate()
{
  ui_.msr_x->setText(QString::number(this->sr_x-x0+xT));
```

```
  ui_.msr_y->setText(QString::number(this->sr_y-y0+yT));
  ui_.msr_z->setText(QString::number(this->sr_z-z0+zT));
  ui_.msr_qx->setText(QString::number(this->sr_qx-qx0+qxT));
  ui_.msr_qy->setText(QString::number(this->sr_qy-qy0+qyT));
  ui_.msr_qz->setText(QString::number(this->sr_qz-qz0+qzT));
  ui_.tsr_x->setText(QString::number(this->sr_x-x0+xT));
  ui_.tsr_y->setText(QString::number(this->sr_y-y0+xT));
  ui_.tsr_z->setText(QString::number(this->sr_z-z0+zT));
  ui_.tsr_qx->setText(QString::number(this->sr_qx-qx0+qxT));
  ui_.tsr_qy->setText(QString::number(this->sr_qy-qy0+qyT));
  ui_.tsr_qz->setText(QString::number(this->sr_qz-qz0+qzT));
}

void Delta_haptic_device::setJoints()
{
  std::vector<double> angles(6);
  angles[0] = ui_.q1DsB->value()*pi/180;
  angles[1] = ui_.q2DsB->value()*pi/180;
  angles[2] = ui_.q3DsB->value()*pi/180;
  angles[3] = ui_.q4DsB->value()*pi/180;
  angles[4] = ui_.q5DsB->value()*pi/180;
  angles[5] = ui_.q6DsB->value()*pi/180;
  move_in_joints_srv_.request.positions = angles;
  move_in_joints_srv_.request.velocity = man_nom_vel/100;
  move_in_joints_srv_.request.acceleration = 0.1;

  move_in_joints_client_.call(move_in_joints_srv_);
}

void Delta_haptic_device::EqualJoints()
{
  ui_.q1DsB->setValue(ui_.msr_q1->toPlainText().toFloat());
  ui_.q2DsB->setValue(ui_.msr_q2->toPlainText().toFloat());
  ui_.q3DsB->setValue(ui_.msr_q3->toPlainText().toFloat());
  ui_.q4DsB->setValue(ui_.msr_q4->toPlainText().toFloat());
  ui_.q5DsB->setValue(ui_.msr_q5->toPlainText().toFloat());
  ui_.q6DsB->setValue(ui_.msr_q6->toPlainText().toFloat());
}

void Delta_haptic_device::setCart()
{
  move_in_cart_srv_.request.pose.pose.position.x = ui_.xWDsB->value()+x0-xT;
  move_in_cart_srv_.request.pose.pose.position.y = ui_.yWDsB->value()+y0-yT;
  move_in_cart_srv_.request.pose.pose.position.z = ui_.zWDsB->value()+z0-zT;
  move_in_cart_srv_.request.pose.pose.orientation.x = sin((ui_.qxWDsB-
>value()+qx0-qxT)*pi/360)*cos((ui_.qyWDsB->value()+qy0-
qyT)*pi/360)*cos((ui_.qzWDsB->value()+qz0-qzT)*pi/360)-cos((ui_.qxWDsB-
>value()+qx0-qxT)*pi/360)*sin((ui_.qyWDsB->value()+qy0-
qyT)*pi/360)*sin((ui_.qzWDsB->value()+qz0-qzT)*pi/360);
  move_in_cart_srv_.request.pose.pose.orientation.y = cos((ui_.qxWDsB-
>value()+qx0-qxT)*pi/360)*sin((ui_.qyWDsB->value()+qy0-
qyT)*pi/360)*cos((ui_.qzWDsB->value()+qz0-qzT)*pi/360)-sin((ui_.qxWDsB-
>value()+qx0-qxT)*pi/360)*cos((ui_.qyWDsB->value()+qy0-
qyT)*pi/360)*sin((ui_.qzWDsB->value()+qz0-qzT)*pi/360);
```

```
  move_in_cart_srv_.request.pose.pose.orientation.z = cos((ui_.qxWDsB-
>value()+qx0-qxT)*pi/360)*cos((ui_.qyWDsB->value()+qy0-
qyT)*pi/360)*sin((ui_.qzWDsB->value()+qz0-qzT)*pi/360)-sin((ui_.qxWDsB-
>value()+qx0-qxT)*pi/360)*sin((ui_.qyWDsB->value()+qy0-
qyT)*pi/360)*cos((ui_.qzWDsB->value()+qz0-qzT)*pi/360);
  move_in_cart_srv_.request.pose.pose.orientation.w = cos((ui_.qxWDsB-
>value()+qx0-qxT)*pi/360)*cos((ui_.qyWDsB->value()+qy0-
qyT)*pi/360)*cos((ui_.qzWDsB->value()+qz0-qzT)*pi/360)-sin((ui_.qxWDsB-
>value()+qx0-qxT)*pi/360)*sin((ui_.qyWDsB->value()+qy0-
qyT)*pi/360)*sin((ui_.qzWDsB->value()+qz0-qzT)*pi/360);

  move_in_cart_srv_.request.velocity = this->man_nom_vel/100;
  move_in_cart_srv_.request.acceleration = 0.1;

  move_in_cart_client_.call(move_in_cart_srv_);
}

void Delta_haptic_device::EqualCart()
{
  ui_.xWDsB->setValue(ui_.msr_x->toPlainText().toFloat());
  ui_.yWDsB->setValue(ui_.msr_y->toPlainText().toFloat());
  ui_.zWDsB->setValue(ui_.msr_z->toPlainText().toFloat());
  ui_.qxWDsB->setValue(ui_.msr_qx->toPlainText().toFloat());
  ui_.qyWDsB->setValue(ui_.msr_qy->toPlainText().toFloat());
  ui_.qzWDsB->setValue(ui_.msr_qz->toPlainText().toFloat());
}

void Delta_haptic_device::setNewOrigin()
{
  this->x0 = ui_.x0DsB->value();
  this->y0 = ui_.y0DsB->value();
  this->z0 = ui_.z0DsB->value();
  this->qx0 = ui_.qx0DsB->value();
  this->qy0 = ui_.qy0DsB->value();
  this->qz0 = ui_.qz0DsB->value();

  ui_.msr_x0->setText(QString::number(this->x0));
  ui_.msr_y0->setText(QString::number(this->y0));
  ui_.msr_z0->setText(QString::number(this->z0));
  ui_.msr_qx0->setText(QString::number(this->qx0));
  ui_.msr_qy0->setText(QString::number(this->qy0));
  ui_.msr_qz0->setText(QString::number(this->qz0));

  ui_.msr_x->setText(QString::number(this->sr_x-x0+xT));
  ui_.msr_y->setText(QString::number(this->sr_y-y0+yT));
  ui_.msr_z->setText(QString::number(this->sr_z-z0+zT));
  ui_.msr_qx->setText(QString::number(this->sr_qx-qx0+qxT));
  ui_.msr_qy->setText(QString::number(this->sr_qy-qy0+qyT));
  ui_.msr_qz->setText(QString::number(this->sr_qz-qz0+qzT));
  ui_.tsr_x->setText(QString::number(this->sr_x-x0+xT));
  ui_.tsr_y->setText(QString::number(this->sr_y-y0+xT));
  ui_.tsr_z->setText(QString::number(this->sr_z-z0+zT));
  ui_.tsr_qx->setText(QString::number(this->sr_qx-qx0+qxT));
  ui_.tsr_qy->setText(QString::number(this->sr_qy-qy0+qyT));
  ui_.tsr_qz->setText(QString::number(this->sr_qz-qz0+qzT));
```

```
}

void Delta_haptic_device::setNewTool()
{
  this->xT = ui_.xTDsB->value();
  this->yT = ui_.yTDsB->value();
  this->zT = ui_.zTDsB->value();
  this->qxT = ui_.qxTDsB->value();
  this->qyT = ui_.qyTDsB->value();
  this->qzT = ui_.qzTDsB->value();

  ui_.msr_x->setText(QString::number(this->sr_x-x0+xT));
  ui_.msr_y->setText(QString::number(this->sr_y-y0+yT));
  ui_.msr_z->setText(QString::number(this->sr_z-z0+zT));
  ui_.msr_qx->setText(QString::number(this->sr_qx-qx0+qxT));
  ui_.msr_qy->setText(QString::number(this->sr_qy-qy0+qyT));
  ui_.msr_qz->setText(QString::number(this->sr_qz-qz0+qzT));
  ui_.tsr_x->setText(QString::number(this->sr_x-x0+xT));
  ui_.tsr_y->setText(QString::number(this->sr_y-y0+xT));
  ui_.tsr_z->setText(QString::number(this->sr_z-z0+zT));
  ui_.tsr_qx->setText(QString::number(this->sr_qx-qx0+qxT));
  ui_.tsr_qy->setText(QString::number(this->sr_qy-qy0+qyT));
  ui_.tsr_qz->setText(QString::number(this->sr_qz-qz0+qzT));
}

void Delta_haptic_device::setDSF()
{
  this->dsf_Float64_msg_.data = ui_.DSF->value();
  this->dsf_publisher_.publish(this->dsf_Float64_msg_);
}

void Delta_haptic_device::setASF()
{
  this->asf_Float64_msg_.data = ui_.ASF->value();
  this->asf_publisher_.publish(this->asf_Float64_msg_);
}

void Delta_haptic_device::setState(bool state)
{
  this->state_Bool_msg_.data = state;
  this->state_publisher_.publish(this->state_Bool_msg_);
}

void Delta_haptic_device::setForceFeedback(bool force_feedback)
{
  this->force_feedback_Bool_msg_.data = force_feedback;
  this->force_feedback_publisher_.publish(this->force_feedback_Bool_msg_);
}

void Delta_haptic_device::setTorqueFeedback(bool torque_feedback)
{
  this->torque_feedback_Bool_msg_.data = torque_feedback;
  this->torque_feedback_publisher_.publish(this->torque_feedback_Bool_msg_);
}
```

```
void Delta_haptic_device::storeData()
{

  std::string file_name;
    if (ui_.FileName->toPlainText() != "")
      file_name = (ui_.FileName->toPlainText()).toStdString();
    else
      file_name = "test";

    bag.open(file_name+".bag", rosbag::bagmode::Write);

  store = true;
  emit storing(true);
}

void Delta_haptic_device::stopDataStoring()
{
  bag.close();
  store = false;
  emit storing(false);
}


void Delta_haptic_device::LogUpdate(QString msg)
{
  if (show)
  {
    QString text = type + ' ' + LogMessage;
    QString sentence = tr("<font color='%1'>%2</font>");
    ui_.LogArea->append(sentence.arg(colour, text));
  }
}

} // namespace
PLUGINLIB_DECLARE_CLASS(iri_gui_delta_haptic_device, Delta_haptic_device,
iri_gui_delta_haptic_device::Delta_haptic_device, rqt_gui_cpp::Plugin)
```

5. Annex E: Manual d'utilització del sistema

5.1. Instal·lació

Per fer servir tot el sistema, en primer lloc l'usuari haurà de descarregar, compilar i instal·lar els drivers necessaris del repositori de l'IRI. Per fer-ho, cal accedir al terminal i escriure:

```
$ mkdir iri-lab
$ cd iri-lab
$ svn checkout https://devel.iri.upc.edu/labrobotica/ --depth
immediates
$ cd labrobotica
$ svn update --set-depth infinity algorithms/ drivers/ tools/
```

També cal obtenir la carpeta sdk-3.5.3, que per motius de llicències no pot ser pujada al repositori de l'IRI. S'ha de demanar a algun dels encarregats del laboratori de l'IRI.

Un cop aconseguida aquesta carpeta, cal modificar l'arxiu del driver anomenat "CmakeLists.txt" situat a ~/iri-lab/labrobotica/drivers/delta_haptic_device/trunk/src.

Els apartats INCLUDE_DIRECTORIES i TARGET_LINK_LIBRARIES s'han de modificar amb el path personalitzat del sistema on s'ha desat la carpeta sdk-3.5.3.

Un cop fet això, es pot procedir a compilar i instal·lar els drivers:

```
$ cd ~/iri-lab/labrobotica/drivers/delta_haptic_device/trunk/build
$ cmake ..
$ make
$ sudo make install
$ cd ~/iri-lab/labrobotica/drivers/ftc_force_sensor/trunk/build
$ cmake ..
$ make
$ sudo make install
$ cd ~/iri-lab/labrobotica/drivers/staubli_robot/trunk/build
$ cmake ..
$ make
$ sudo make install
```

Un cop s'hagin instal·lat aquests drivers, es pot procedir a crear l'entorn de treball de ROS:

```
$ mkdir -p ~/iri-lab/iri_ws/src
$ cd ~/iri-lab/iri_ws/src
$ catkin_init_workspace
$ cd ~/iri-lab/iri_ws
$ catkin_make
$ echo "source ~/iri-lab/iri_ws/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
$ cd ~/iri-lab/iri_ws/src
$ wstool init
$ wstool update
$ roscd
$ cd ..
$ catkin_make
```

Seguidament, s'han de descarregar els packages fets servir:

```
$ roscd
$ cd ../src
$ wstool set iri_core --svn
https://devel.iri.upc.edu/labrobotica/ros/iri-ros-
pkg_hydro/metapackages/iri_core
$ wstool set iri_delta_haptic_device --svn
https://devel.iri.upc.edu/labrobotica/ros/iri-ros-
pkg_hydro/metapackages/iri_common_drivers/iri_delta_haptic_dev
ice
$ wstool set iri_ftc_force_sensor --svn
https://devel.iri.upc.edu/labrobotica/ros/iri-ros-
pkg_hydro/metapackages/iri_commondrivers/iri_ftc_force_sensor
$ wstool set iri_staubli --svn
https://devel.iri.upc.edu/labrobotica/ros/iri-ros-
pkg_hydro/metapackages/iri_staubli
$ wstool set iri_gui_delta_haptic_device --svn
https://devel.iri.upc.edu/labrobotica/ros/iri-ros-
pkg_hydro/metapackages/iri_gui_delta_haptic_device
$ wstool update
$ cd ..
$ catkin_make
```

5.2. Utilització

Aquest manual suposa que la posada en marxa detallada a l'apartat 6 del document de la Memòria del projecte ha estat realitzada. En cas de no haver-la fet, cal realitzar-la per habilitar el funcionament dels diferents dispositius.

Per procedir a la utilització dels nodes, el primer és obrir un terminal i escriure:

```
$ roscore
```

Això habilita el funcionament de ROS.

Seguidament, es poden iniciar tots els nodes que formen el sistema (l'ordre és arbitrari):

- Node del dispositiu hàptic:

```
$ rosrun iri_delta_haptic_device iri_delta_haptic_device
```

- Node del sensor de forces:

```
$ rosrun iri_ftc_force_sensor iri_ftc_force_sensor
```

- Node del braç robòtic:

```
$ roslaunch iri_staubli_controller iri_staubli_controller.launch
IP:=192.168.100.232
```

- Node de l'aplicació gràfica:

```
$ rqt
```

I seleccionar a les comandes interactives Plugins, Topics, Delta_Haptic_Device.

Un cop fet això, tots els nodes ja estaran operatius per a ser fet servits a la pestanya de Telecommunication. Si, al contrari, es vol treballar a la pestanya Manual, cal deixar operatiu només el node de l'element que es vol fer servir. Per sortir d'un node, cal anar al terminal on s'està executant i pressionant la combinació de tecles Ctrl+C.

Durant l'operació de telecomunicació l'usuari pot veure per pantalla, gràcies a l'aplicació gràfica, les diferents dades que recol·lecta cada dispositiu i els avisos que els nodes envien (amb la zona anomenada Event Log).

Si l'usuari vol guardar les dades que s'estan enviant, primer ha de seleccionar quines dades vol guardar i després prémer el botó Start DataStoring. Es guardaran totes les dades seleccionades fins al moment en què l'usuari pressioni el botó Stop DataStoring. Al directori del workspace de ROS, apareixerà un fitxer d'extensió ".bag" amb les dades guardades.