Sistema de monitorización inercial del movimiento de las extremidades superiores - Código fuente

Daniel Fernández Villanueva

22 de julio de 2013

Índice general

Parte I

Código fuente

Capítulo 1

PAQUETE xsens_driver

1.1. xsens_node.cpp

```
/*
 * Programa que publica en topics de ROS
 * los datos obtenidos de los sensores Xsens
 *
 * Autor: Daniel Fernández Villanueva
 * Mayo de 2013
 *
 */

#include <iostream>
#include <cmath>
#include <xsens_driver/xsens_driver.h>
#include <ros/ros.h>
#include <geometry_msgs/Vector3Stamped.h>
#include <geometry_msgs/QuaternionStamped.h>
#include <std_msgs/Float64MultiArray.h>
#include <dfv/dfv.h>
#include <xsens_driver/utils.h>

int main(int argc, char** argv)
{
    // Declaración de un objeto driver.
    // Valores por defecto:
    // OutputMode: CMT_OUTPUTMODE_CALIB | CMT_OUTPUTMODE_ORIENT
    // OutputSettings: CMT_OUTPUTSETTINGS_ORIENTMODE_QUATERNION
    xsens::Driver driver;

    // Aquí podemos cambiar la configuración del sensor

    // Número de dispositivos detectados (sin contar el Xbus Master)
    ROS_INFO("Detected sensor count: %d", driver.GetMtCount());

    // Asignamos a los sensores una matriz de pre-rotación unitaria
    for(unsigned int i = 0; i < driver.GetMtCount(); ++i)
    {
        driver.SetAlignmentMatrix(i, xsens::DfvToCmtMatrix(dfv::Matrix::Identity(3)));
    }

    // Ejemplo para cambiar el modo de salida del sensor
    // para que nos de la matriz de rotación en lugar
```

```
41      // del cuaternión de orientación:
42
43      //driver.SetOutputSettings(CMT_OUTPUTSETTINGS_ORIENTMODE_MATRIX);
44
45      // Inicializamos el driver. Esto realizará la configuración del sensor
46      // con los valores que le hayamos asignado hasta ahora
47      // y lo pondrá en modo de medida
48      if(driver.Initialize() == false)
49      {
50          std::cout << "ERROR: No Xsens IMUs found. Quitting..." << std::endl;
51          return -1;
52      }
53
54      // Inicialización de ROS
55      ROS_INFO("Initializing ROS...");
56      ros::init(argc, argv, "xsens_node");
57      ros::NodeHandle node_handle("~");
58
59      // Asignamos valor a algunos parámetros
60      node_handle.setParam("sensor_count", (int)driver.GetMtCount());
61      node_handle.setParam("output_mode", (int)driver.GetOutputMode());
62      node_handle.setParam("output_settings", (int)driver.GetOutputSettings());
63
64      // Creamos un NodeHandle para cada sensor
65      std::vector<ros::NodeHandle> sensor_node_handles(driver.GetMtCount());
66      for(unsigned int i = 0; i < driver.GetMtCount(); i++)
67      {
68          std::stringstream ss;
69          ss << "sensor" << i;
70          sensor_node_handles[i] = ros::NodeHandle(node_handle, ss.str());
71      }
72
73
74      // Declaramos los publicadores
75      std::vector<ros::Publisher> acc_publishers(driver.GetMtCount());
76      std::vector<ros::Publisher> gyr_publishers(driver.GetMtCount());
77      std::vector<ros::Publisher> mag_publishers(driver.GetMtCount());
78
79      std::vector<ros::Publisher> raw_acc_publishers(driver.GetMtCount());
80      std::vector<ros::Publisher> raw_gyr_publishers(driver.GetMtCount());
81      std::vector<ros::Publisher> raw_mag_publishers(driver.GetMtCount());
82
83      std::vector<ros::Publisher> ori_quat_publishers(driver.GetMtCount());
84      std::vector<ros::Publisher> ori_matrix_publishers(driver.GetMtCount());
85      std::vector<ros::Publisher> ori_euler_publishers(driver.GetMtCount());
86
87      std::vector<ros::Publisher> pos_lla_publishers(driver.GetMtCount());
88
89      std::vector<ros::Publisher> gps_llh_publishers(driver.GetMtCount());
90      std::vector<ros::Publisher> gps_vel_publishers(driver.GetMtCount());
91
92      // Creamos los topics a publicar
93      for(unsigned int i = 0; i < driver.GetMtCount(); i++)
94      {
95          // Datos calibrados
96          if((driver.GetOutputMode() & CMT_OUTPUTMODE_CALIB) != 0)
97          {
```

```
98              acc_publishers[i] = sensor_node_handles[i].advertise<geometry_msgs::
                    Vector3Stamped>("acc", 1000);
99              gyr_publishers[i] = sensor_node_handles[i].advertise<geometry_msgs::
                    Vector3Stamped>("gyr", 1000);
100             mag_publishers[i] = sensor_node_handles[i].advertise<geometry_msgs::
                    Vector3Stamped>("mag", 1000);
101         }
102
103         // Datos crudos
104         if((driver.GetOutputMode() & CMT_OUTPUTMODE_RAW) != 0)
105         {
106             raw_acc_publishers[i] = sensor_node_handles[i].advertise<geometry_msgs::
                    Vector3Stamped>("raw_acc", 1000);
107             raw_gyr_publishers[i] = sensor_node_handles[i].advertise<geometry_msgs::
                    Vector3Stamped>("raw_gyr", 1000);
108             raw_mag_publishers[i] = sensor_node_handles[i].advertise<geometry_msgs::
                    Vector3Stamped>("raw_mag", 1000);
109         }
110
111         if((driver.GetOutputMode() & CMT_OUTPUTMODE_POSITION) != 0)
112         {
113             pos_lla_publishers[i] = sensor_node_handles[i].advertise<geometry_msgs::
                    Vector3Stamped>("pos_lla", 1000);
114         }
115
116         if((driver.GetOutputMode() & CMT_OUTPUTMODE_GPSPVT_PRESSURE) != 0)
117         {
118             gps_llh_publishers[i] = sensor_node_handles[i].advertise<geometry_msgs::
                    Vector3Stamped>("gps_llh", 1000);
119             gps_vel_publishers[i] = sensor_node_handles[i].advertise<geometry_msgs::
                    Vector3Stamped>("gps_vel", 1000);
120         }
121
122         // Datos de orientación
123         if((driver.GetOutputMode() & CMT_OUTPUTMODE_ORIENT) != 0)
124         {
125             // Cuaternión de orientación
126             if((driver.GetOutputSettings() & CMT_OUTPUTSETTINGS_ORIENTMODE_MASK) ==
                    CMT_OUTPUTSETTINGS_ORIENTMODE_QUATERNION)
127             {
128                 ori_quat_publishers[i] = sensor_node_handles[i].advertise<geometry_msgs
                        ::QuaternionStamped>("ori_quat", 1000);
129             }
130
131             // Matriz de orientación
132             if((driver.GetOutputSettings() & CMT_OUTPUTSETTINGS_ORIENTMODE_MASK) ==
                    CMT_OUTPUTSETTINGS_ORIENTMODE_MATRIX)
133             {
134                 ori_matrix_publishers[i] = sensor_node_handles[i].advertise<std_msgs::
                        Float64MultiArray>("ori_matrix", 1000);
135             }
136
137             // Ángulos de Euler
138             if((driver.GetOutputSettings() & CMT_OUTPUTSETTINGS_ORIENTMODE_MASK) ==
                    CMT_OUTPUTSETTINGS_ORIENTMODE_EULER)
139             {
140                 ori_euler_publishers[i] = sensor_node_handles[i].advertise<std_msgs::
                        Float64MultiArray>("ori_euler", 1000);
```

```
141             }
142         }
143     }
144 
145     // Contador
146     int count = 0;
147 
148     // Empezamos a publicar los datos
149     ROS_INFO("Now publishing data...");
150 
151     while(driver.SpinOnce() && ros::ok())
152     {
153         for(unsigned int i = 0; i < driver.GetMtCount(); i++)
154         {
155             if((driver.GetOutputMode() & CMT_OUTPUTMODE_CALIB) != 0)
156             {
157                 geometry_msgs::Vector3Stamped msg;
158 
159                 msg = xsens::ToVector3StampedMsg(xsens::CmtToDfvVector(driver.GetCalData
                        (i).m_acc));
160                 msg.header.seq = count;
161                 acc_publishers[i].publish(msg);
162 
163                 msg = xsens::ToVector3StampedMsg(xsens::CmtToDfvVector(driver.GetCalData
                        (i).m_gyr));
164                 msg.header.seq = count;
165                 gyr_publishers[i].publish(msg);
166 
167                 msg = xsens::ToVector3StampedMsg(xsens::CmtToDfvVector(driver.GetCalData
                        (i).m_mag));
168                 msg.header.seq = count;
169                 mag_publishers[i].publish(msg);
170             }
171 
172             if((driver.GetOutputMode() & CMT_OUTPUTMODE_RAW) != 0)
173             {
174                 geometry_msgs::Vector3Stamped msg;
175 
176                 msg = xsens::ToVector3StampedMsg(xsens::CmtToDfvShortVector(driver.
                        GetRawData(i).m_acc));
177                 msg.header.seq = count;
178                 raw_acc_publishers[i].publish(msg);
179 
180                 msg = xsens::ToVector3StampedMsg(xsens::CmtToDfvShortVector(driver.
                        GetRawData(i).m_gyr));
181                 msg.header.seq = count;
182                 raw_gyr_publishers[i].publish(msg);
183 
184                 msg = xsens::ToVector3StampedMsg(xsens::CmtToDfvShortVector(driver.
                        GetRawData(i).m_mag));
185                 msg.header.seq = count;
186                 raw_mag_publishers[i].publish(msg);
187             }
188 
189             if((driver.GetOutputMode() & CMT_OUTPUTMODE_POSITION) != 0)
190             {
191                 geometry_msgs::Vector3Stamped msg;
192 
```

```
193                 msg = xsens::ToVector3StampedMsg(xsens::CmtToDfvVector(driver.
                        GetPositionLLA(i)));
194                 msg.header.seq = count;
195                 pos_lla_publishers[i].publish(msg);
196             }
197
198             if((driver.GetOutputMode() & CMT_OUTPUTMODE_GPSPVT_PRESSURE) != 0)
199             {
200                 CmtGpsPvtData data = driver.GetGpsPvtData(i);
201
202                 geometry_msgs::Vector3Stamped llh_msg;
203                 llh_msg.vector.x = (float)data.m_latitude;
204                 llh_msg.vector.y = (float)data.m_longitude;
205                 llh_msg.vector.z = (float)data.m_height;
206                 llh_msg.header.stamp = ros::Time::now();
207                 llh_msg.header.seq = count;
208                 gps_llh_publishers[i].publish(llh_msg);
209
210                 geometry_msgs::Vector3Stamped vel_msg;
211                 vel_msg.vector.x = (float)data.m_veln;
212                 vel_msg.vector.y = (float)data.m_vele;
213                 vel_msg.vector.z = (float)data.m_veld;
214                 vel_msg.header.stamp = ros::Time::now();
215                 vel_msg.header.seq = count;
216                 gps_vel_publishers[i].publish(vel_msg);
217
218             }
219
220             // Datos de orientación
221             if((driver.GetOutputMode() & CMT_OUTPUTMODE_ORIENT) != 0)
222             {
223                 // Cuaternión de orientación
224                 if((driver.GetOutputSettings() & CMT_OUTPUTSETTINGS_ORIENTMODE_MASK) ==
                        CMT_OUTPUTSETTINGS_ORIENTMODE_QUATERNION)
225                 {
226                     geometry_msgs::QuaternionStamped msg;
227                     msg.quaternion.w = driver.GetOriQuat(i).m_data[0];
228                     msg.quaternion.x = driver.GetOriQuat(i).m_data[1];
229                     msg.quaternion.y = driver.GetOriQuat(i).m_data[2];
230                     msg.quaternion.z = driver.GetOriQuat(i).m_data[3];
231                     msg.header.seq = count;
232                     msg.header.stamp = ros::Time::now();
233                     ori_quat_publishers[i].publish(msg);
234                 }
235
236                 // Matriz de orientación
237                 if((driver.GetOutputSettings() & CMT_OUTPUTSETTINGS_ORIENTMODE_MASK) ==
                        CMT_OUTPUTSETTINGS_ORIENTMODE_MATRIX)
238                 {
239                     std_msgs::Float64MultiArray msg;
240                     msg.data.clear();
241                     msg.data.resize(9);
242                     msg.data[0] = driver.GetOriMatrix(i).m_data[0][0];
243                     msg.data[1] = driver.GetOriMatrix(i).m_data[0][1];
244                     msg.data[2] = driver.GetOriMatrix(i).m_data[0][2];
245                     msg.data[3] = driver.GetOriMatrix(i).m_data[1][0];
246                     msg.data[4] = driver.GetOriMatrix(i).m_data[1][1];
247                     msg.data[5] = driver.GetOriMatrix(i).m_data[1][2];
```

```
                    msg.data[6] = driver.GetOriMatrix(i).m_data[2][0];
                    msg.data[7] = driver.GetOriMatrix(i).m_data[2][1];
                    msg.data[8] = driver.GetOriMatrix(i).m_data[2][2];
                    ori_matrix_publishers[i].publish(msg);
                }

                // Ángulos de Euler
                if((driver.GetOutputSettings() & CMT_OUTPUTSETTINGS_ORIENTMODE_MASK) ==
                    CMT_OUTPUTSETTINGS_ORIENTMODE_EULER)
                {
                    std_msgs::Float64MultiArray msg;
                    msg.data.clear();
                    msg.data.resize(3);
                    msg.data[0] = driver.GetOriEuler(i).m_roll;
                    msg.data[1] = driver.GetOriEuler(i).m_pitch;
                    msg.data[2] = driver.GetOriEuler(i).m_yaw;
                    ori_euler_publishers[i].publish(msg);
                }
            }
        }

        ++count;
        ros::spinOnce();
        //ros::Duration(0.1).sleep();


    }

    ROS_INFO("Finishing program...");

    return 0;
}
```

1.2. xsens_driver.h

```
/*
 * Clase Driver encargada de la configuración
 * y toma de datos de los sensores Xsens
 *
 * Autor: Daniel Fernández Villanueva
 * Mayo de 2013
 *
 */

#ifndef XSENS_DRIVER_H
#define XSENS_DRIVER_H

#include <vector>
#include <sstream>

#include <unistd.h>
#include <sys/ioctl.h>
#include <fcntl.h>

#include <ros/ros.h>

#include <xsens_driver/cmtdef.h>
#include <xsens_driver/xsens_time.h>
#include <xsens_driver/xsens_list.h>
#include <xsens_driver/cmtscan.h>
#include <xsens_driver/cmt3.h>
#include <xsens_driver/xsens_sensor.h>

namespace xsens
{

    class Driver
    {
        public:
            Driver();
            ~Driver();

            bool Initialize();

            void SetOutputMode(CmtOutputMode output_mode);
            CmtOutputMode GetOutputMode() const;

            void SetOutputSettings(CmtOutputSettings output_settings);
            CmtOutputSettings GetOutputSettings() const;

            void SetAlignmentMatrix(unsigned int sensor_index, CmtMatrix
                alignment_matrix);

            bool SpinOnce();
            bool RetrieveData();
            unsigned int GetMtCount();
            CmtOutputMode GetOutputMode();
            CmtOutputSettings GetOutputSettings();

            // Funciones para obtener datos
```

```
55              CmtQuat&       GetOriQuat(int mt_index = 0);
56              CmtMatrix&     GetOriMatrix(int mt_index = 0);
57              CmtEuler&      GetOriEuler(int mt_index = 0);
58              CmtRawData&    GetRawData(int mt_index = 0);
59              CmtCalData&    GetCalData(int mt_index = 0);
60
61              // Funciones para implementar en el futuro
62              CmtVector&     GetPositionLLA(int mt_index = 0);
63              CmtGpsPvtData& GetGpsPvtData(int mt_index = 0);
64
65              // Vector de sensores
66              std::vector<Sensor> v_sensors;
67
68          private:
69              Cmt3               cmt3;
70              unsigned int       mt_count;
71              CmtOutputMode      output_mode;
72              CmtOutputSettings  output_settings;
73              short              skip_factor;
74              short              skip_factor_count;
75
76              Packet* lp_packet;
77
78              unsigned short        sample_data;
79
80              bool DoHardwareScan();
81              bool SetConfiguration();
82
83      };
84  };
85
86  #endif
```

1.3. xsens_driver.cpp

```cpp
#include <xsens_driver/xsens_driver.h>

namespace xsens
{

    Driver::Driver():
        mt_count(0),
        output_mode(CMT_OUTPUTMODE_CALIB | CMT_OUTPUTMODE_ORIENT),
        output_settings(CMT_OUTPUTSETTINGS_ORIENTMODE_QUATERNION |
            CMT_OUTPUTSETTINGS_TIMESTAMP_SAMPLECNT),
        skip_factor(10),
        skip_factor_count(0),
        lp_packet(NULL)
    {
        //this->output_settings |= CMT_OUTPUTSETTINGS_TIMESTAMP_SAMPLECNT;
        if(this->DoHardwareScan() == false)
        {
            ROS_ERROR("In function %s at line %d in file %s", __PRETTY_FUNCTION__,
                __LINE__, __FILE__);
            this->cmt3.closePort();
        }
    }

    Driver::~Driver()
    {
        delete this->lp_packet;
        this->cmt3.closePort();
    }

    void Driver::SetOutputMode(CmtOutputMode output_mode)
    {
        this->output_mode = output_mode;
    }

    CmtOutputMode Driver::GetOutputMode() const
    {
        return this->output_mode;
    }

    void Driver::SetOutputSettings(CmtOutputSettings output_settings)
    {
        this->output_settings = (output_settings |
            CMT_OUTPUTSETTINGS_TIMESTAMP_SAMPLECNT);
    }

    CmtOutputSettings Driver::GetOutputSettings() const
    {
        //return (this->output_settings & CMT_OUTPUTSETTINGS_ORIENTMODE_MASK);
        return this->output_settings;
    }

    void Driver::SetAlignmentMatrix(unsigned int sensor_index, CmtMatrix
        alignment_matrix)
    {
        this->v_sensors[sensor_index].alignment_matrix = alignment_matrix;
```

```
52      }
53
54      bool Driver::DoHardwareScan()
55      {
56          XsensResultValue  res;
57          List<CmtPortInfo> port_info;
58          unsigned long     port_count = 0;
59
60          ROS_INFO("Scanning for connected Xsens devices...");
61          xsens::cmtScanPorts(port_info);
62          port_count = port_info.length();
63          ROS_INFO("Scanning done");
64
65          if (port_count == 0)
66          {
67              ROS_ERROR("No motion trackers found");
68              return false;
69          }
70
71          for (int i = 0; i < (int)port_count; i++)
72          {
73              std::stringstream ss;
74              ss << "Using COM port " << port_info[i].m_portName << " at ";
75              switch (port_info[i].m_baudrate)
76              {
77                  case B9600:
78                      ss << "9k6";
79                      break;
80                  case B19200:
81                      ss << "19k2";
82                      break;
83                  case B38400:
84                      ss << "38k4";
85                      break;
86                  case B57600:
87                      ss << "57k6";
88                      break;
89                  case B115200:
90                      ss << "115k2";
91                      break;
92                  case B230400:
93                      ss << "230k4";
94                      break;
95                  case B460800:
96                      ss << "460k8";
97                      break;
98                  case B921600:
99                      ss << "921k6";
100                     break;
101                 default:
102                     ss << port_info[i].m_baudrate;
103             }
104             ss << " baud" << std::endl;
105             ROS_INFO("%s", ss.str().c_str());
106         }
107         ROS_INFO("Opening ports...");
108
```

```
109         // open the port which the device is connected to and connect at the device's
                baudrate.
110
111         for (int p = 0; p < (int)port_count; p++)
112         {
113             res = this->cmt3.openPort(port_info[p].m_portName,
114                                       port_info[p].m_baudrate);
115             if (res != XRV_OK)
116             {
117                 ROS_ERROR("In function %s at line %d in file %s", __PRETTY_FUNCTION__,
                        __LINE__, __FILE__);
118                 return false;
119             }
120         }
121         std::cout << "Done" << std::endl;
122
123         // set the measurement timeout to 100 ms (default is 16 ms)
124
125         int timeout = 100;
126         res = this->cmt3.setTimeoutMeasurement(timeout);
127         if (res != XRV_OK)
128         {
129             ROS_ERROR("In function %s at line %d in file %s", __PRETTY_FUNCTION__,
                    __LINE__, __FILE__);
130             return false;
131         }
132         ROS_INFO("Timeout set to %d ms", timeout);
133
134         // get the MT sensor count
135
136         ROS_INFO("Retrieving MT count (excluding attached Xbus Master(s))");
137         this->mt_count = this->cmt3.getMtCount();
138         ROS_INFO("MT count: %d", this->mt_count);
139
140         this->v_sensors.resize(this->mt_count);
141
142         // retrieve the device IDs
143
144         ROS_INFO("Retrieving MT device IDs");
145         for (unsigned int j = 0; j < this->mt_count; j++)
146         {
147             // res = this->cmt3.getDeviceId((unsigned char)(j+1), this->device_ids[j]);
148             res = this->cmt3.getDeviceId((unsigned char)(j+1), this->v_sensors[j].
                    device_id);
149             if (res != XRV_OK)
150             {
151                 ROS_ERROR("In function %s at line %d in file %s", __PRETTY_FUNCTION__,
                        __LINE__, __FILE__);
152                 return false;
153             }
154         }
155
156         return true;
157     }
158
159     bool Driver::SetConfiguration()
160     {
161         XsensResultValue res;
```

```
162
163         // set the sensor to config state
164
165         res = this->cmt3.gotoConfig();
166         if (res != XRV_OK)
167         {
168             ROS_ERROR("In function %s at line %d in file %s", __PRETTY_FUNCTION__,
                    __LINE__, __FILE__);
169             ROS_ERROR("Could not go to configuration mode");
170             return false;
171         }
172
173         unsigned short sample_freq;
174         sample_freq = this->cmt3.getSampleFrequency();
175
176         // set the device output mode for the devices
177
178         if ((this->output_mode & CMT_OUTPUTMODE_ORIENT) == 0)
179         {
180             this->output_settings = 0;
181             this->output_settings |= CMT_OUTPUTSETTINGS_TIMESTAMP_SAMPLECNT;
182         }
183
184         ROS_INFO("Configuring your mode selection");
185         for (unsigned int i = 0; i < this->mt_count; i++)
186         {
187             CmtDeviceMode device_mode(this->output_mode,
188                                       this->output_settings,
189                                       sample_freq);
190             if ((this->v_sensors[i].device_id & 0xFFF00000) != 0x00500000)
191             {
192                 // not an MTi-G, remove all GPS related stuff
193                 device_mode.m_outputMode &= 0xFF0F;
194             }
195             res = this->cmt3.setDeviceMode(device_mode, true, this->v_sensors[i].
                    device_id);
196             if (res != XRV_OK)
197             {
198                 ROS_ERROR("In function %s at line %d in file %s", __PRETTY_FUNCTION__,
                        __LINE__, __FILE__);
199                 return false;
200             }
201         }
202
203         // Set aligment Matrix
204         for(unsigned int i = 0; i < this->mt_count; i++)
205         {
206             if(this->cmt3.setObjectAlignmentMatrix(this->v_sensors[i].alignment_matrix,
                     this->v_sensors[i].device_id) != XRV_OK)
207             {
208                 ROS_ERROR("Could not set alignment matrix for object %d", i);
209                 return false;
210             }
211             else
212             {
213                 ROS_INFO("Alignment matrix set for object %d to M", i);
214             }
215         }
```

```
216
217         res = this->cmt3.gotoMeasurement();
218         if (res != XRV_OK)
219         {
220             ROS_ERROR("ERROR: go to measurement");
221             return false;
222         }
223
224         return true;
225     }
226
227     bool Driver::Initialize()
228     {
229         /*if (this->DoHardwareScan() == false)
230         {
231             std::cout << "ERROR: DoHardwareScan()" << std::endl;
232             this->cmt3.closePort();
233             return false;
234         }*/
235
236         if (this->mt_count == 0)
237         {
238             //ROS_ERROR("In function %s at line %d in file %s", __PRETTY_FUNCTION__,
                    __LINE__, __FILE__);
239             ROS_ERROR("No Imus found.");
240             this->cmt3.closePort();
241             return false;
242         }
243
244         if (this->SetConfiguration() == false)
245         {
246             ROS_ERROR("In function %s at line %d in file %s", __PRETTY_FUNCTION__,
                    __LINE__, __FILE__);
247             return false;
248         }
249
250         this->lp_packet = new Packet((unsigned short)this->mt_count, this->cmt3.isXm())
                ;
251         ROS_INFO("Everything is OK. Retrieving data...");
252
253         return true;
254     }
255
256     bool Driver::SpinOnce()
257     {
258         XsensResultValue res = this->cmt3.waitForDataMessage(this->lp_packet);
259         if (res != XRV_OK)
260         {
261             if ((res == XRV_TIMEOUTNODATA) || (res == XRV_TIMEOUT))
262             {
263                 return true;
264             }
265
266             delete this->lp_packet;
267             this->cmt3.closePort();
268             ROS_ERROR("In function %s at line %d in file %s", __PRETTY_FUNCTION__,
                    __LINE__, __FILE__);
269             return false;
```

```
270         }
271
272         this->sample_data = this->lp_packet->getSampleCounter();
273         if (this->RetrieveData() == false)
274         {
275             //std::cout << "ERROR: RetrieveData()" << std::endl;
276             ROS_ERROR("In function %s at line %d in file %s", __PRETTY_FUNCTION__,
                    __LINE__, __FILE__);
277             return false;
278         }
279
280         return true;
281     }
282
283     bool Driver::RetrieveData()
284     {
285         for (unsigned int i = 0; i < this->mt_count; i++)
286         {
287             if ((this->output_mode & CMT_OUTPUTMODE_RAW) != 0)
288             {
289                 this->v_sensors[i].raw_data.m_acc = this->lp_packet->getRawAcc(i);
290                 this->v_sensors[i].raw_data.m_gyr = this->lp_packet->getRawGyr(i);
291                 this->v_sensors[i].raw_data.m_mag = this->lp_packet->getRawMag(i);
292                 this->v_sensors[i].raw_data.m_temp = this->lp_packet->getRawTemp(i);
293                 continue;
294             }
295
296             if ((this->output_mode & CMT_OUTPUTMODE_TEMP) != 0)
297             {
298                 this->v_sensors[i].temperature_data = this->lp_packet->getTemp(i);
299             }
300
301             if ((this->output_mode & CMT_OUTPUTMODE_CALIB) != 0)
302             {
303                 this->v_sensors[i].calibrated_data = this->lp_packet->getCalData(i);
304             }
305
306             if ((this->output_mode & CMT_OUTPUTMODE_POSITION) != 0)
307             {
308                 if (this->lp_packet->containsPositionLLA(i))
309                 {
310                     //CmtVector positionLLA = this->lp_packet->getPositionLLA();
311                     this->v_sensors[i].position_lla = this->lp_packet->getPositionLLA(i);
312                     /*if (this->result_value != XRV_OK)
313                     {
314                         std::cout << "ERROR: get position LLA" << std::endl;
315                     }*/
316
317                     /*for (int i = 0; i < 2; i++)
318                     {
319                         double deg = positionLLA.m_data[i];
320                         double min = (deg - (int)deg)*60;
321                         double sec = (min - (int)min)*60;
322                     }*/
323                 }
324                 else
325                 {
```

```
326                     ROS_ERROR("In function %s at line %d in file %s",
                            __PRETTY_FUNCTION__, __LINE__, __FILE__);
327                     ROS_ERROR("No PositionLLA data available");
328                 }
329             }
330
331             if((this->output_mode & CMT_OUTPUTMODE_GPSPVT_PRESSURE) != 0)
332             {
333                 if (this->lp_packet->containsGpsPvtData(i))
334                 {
335                     this->v_sensors[i].gps_pvt_data = this->lp_packet->getGpsPvtData(i);
336                     // ROS_INFO("Retrieving GPS pvt Data");
337                 }
338                 else
339                 {
340                     ROS_ERROR("In function %s at line %d in file %s",
                            __PRETTY_FUNCTION__, __LINE__, __FILE__);
341                     ROS_ERROR("No GpsPvt data available");
342                 }
343             }
344
345             if ((this->output_mode & CMT_OUTPUTMODE_ORIENT) == 0)
346             {
347                 continue;
348             }
349
350             switch (this->output_settings & CMT_OUTPUTSETTINGS_ORIENTMODE_MASK)
351             {
352                 case CMT_OUTPUTSETTINGS_ORIENTMODE_QUATERNION:
353                     this->v_sensors[i].quaternion_data = this->lp_packet->getOriQuat(i);
354                     break;
355                 case CMT_OUTPUTSETTINGS_ORIENTMODE_EULER:
356                     this->v_sensors[i].euler_data = this->lp_packet->getOriEuler(i);
357                     break;
358                 case CMT_OUTPUTSETTINGS_ORIENTMODE_MATRIX:
359                     this->v_sensors[i].matrix_data = this->lp_packet->getOriMatrix(i);
360                     break;
361                 default:
362                     break;
363             }
364
365         }
366
367         return true;
368     }
369
370     unsigned int Driver::GetMtCount()
371     {
372         return this->mt_count;
373     }
374
375     CmtOutputMode Driver::GetOutputMode()
376     {
377         return this->output_mode;
378     }
379
380     CmtOutputSettings Driver::GetOutputSettings()
381     {
```

```
382         return this->output_settings;
383     }
384 
385     CmtQuat& Driver::GetOriQuat(int mt_index)
386     {
387         return this->v_sensors[mt_index].quaternion_data;
388     }
389 
390     CmtMatrix& Driver::GetOriMatrix(int mt_index)
391     {
392         return this->v_sensors[mt_index].matrix_data;
393     }
394 
395     CmtEuler& Driver::GetOriEuler(int mt_index)
396     {
397         return this->v_sensors[mt_index].euler_data;
398     }
399 
400     CmtRawData& Driver::GetRawData(int mt_index)
401     {
402         return this->v_sensors[mt_index].raw_data;
403     }
404 
405     CmtCalData& Driver::GetCalData(int mt_index)
406     {
407         return this->v_sensors[mt_index].calibrated_data;
408     }
409 
410     CmtVector& Driver::GetPositionLLA(int mt_index)
411     {
412         return this->v_sensors[mt_index].position_lla;
413     }
414 
415     CmtGpsPvtData& Driver::GetGpsPvtData(int mt_index)
416     {
417         return this->v_sensors[mt_index].gps_pvt_data;
418     }
419 
420 }
```

1.4. xsens_sensor.h

```
/*
 * Clase Sensor en la que la clase Driver almacenará
 * los datos obtenidos por los sensores físicos Xsens,
 * además de ciertos parámetros de configuración
 *
 * Autor: Daniel Fernández Villanueva
 * Mayo de 2013
 *
 */

#ifndef XSENS_SENSOR_H
#define XSENS_SENSOR_H

#include <xsens_driver/cmtdef.h>

namespace xsens
{
    class Sensor
    {
        public:
            Sensor();
            ~Sensor();

            void SetAlignmentMatrix(const CmtMatrix& matrix);

            CmtMatrix         alignment_matrix;

        protected:
        private:
            CmtCalData        calibrated_data;
            CmtQuat           quaternion_data;
            CmtEuler          euler_data;
            CmtMatrix         matrix_data;
            CmtRawData        raw_data;
            CmtVector         position_lla;
            CmtGpsPvtData     gps_pvt_data;
            double            temperature_data;

            CmtDeviceId       device_id;

            friend class Driver;

    };
}

#endif
```

1.5. xsens_sensor.cpp

```
#include <xsens_driver/xsens_sensor.h>

namespace xsens
{
    Sensor::Sensor()
    {
        CmtMatrix matrix;
        matrix.m_data[0][0] = 1.0; matrix.m_data[0][1] = 0.0; matrix.m_data[0][2] =
            0.0;
        matrix.m_data[1][0] = 0.0; matrix.m_data[1][1] = 1.0; matrix.m_data[1][2] =
            0.0;
        matrix.m_data[2][0] = 0.0; matrix.m_data[2][1] = 0.0; matrix.m_data[2][2] =
            1.0;
        this->alignment_matrix = matrix;
    }

    Sensor::~Sensor()
    {
    }

    void Sensor::SetAlignmentMatrix(const CmtMatrix& matrix)
    {
        this->alignment_matrix = matrix;
    }
}
```

1.6. xsens_sensor_subscriber.h

```
/*
 * Clases SensorSubscriber y SensorSubscriberList
 *
 * Estas clases no son utilizadas por el driver.
 * Forman parte de la librería xsens_driver, que
 * proporciona una interfaz sencilla para acceder
 * a los datos publicados en ROS por el driver en
 * otros programas.
 *
 * Autor: Daniel Fernández Villanueva
 * Mayo 2013
 *
 */

#ifndef XSENS_SENSOR_SUBSCRIBER_H
#define XSENS_SENSOR_SUBSCRIBER_H

#include <dfv/dfv.h>
#include <sstream>
#include <xsens_driver/cmtdef.h>
#include <ros/ros.h>
#include <std_msgs/Float64MultiArray.h>
#include <geometry_msgs/Vector3Stamped.h>
#include <geometry_msgs/QuaternionStamped.h>

namespace xsens
{
    class SensorSubscriber
    {
        public:
            SensorSubscriber(unsigned int mt_index_, ros::NodeHandle& node_handle_);
            ~SensorSubscriber();

            bool                    SubscribeToTopics();

            // Función que devuelve el vector aceleración
            const dfv::Vector3    GetAcc() const;

            // Función que devuelve el vector giróscopo
            const dfv::Vector3    GetGyr() const;

            // Función que devuelve el vector campo magnético
            const dfv::Vector3    GetMag() const;

            // Función que devuelve el cuaternión de orientación
            const dfv::Quaternion GetOriQuat() const;

            // Función que devuelve la matriz de orientación
            const dfv::Matrix     GetOriMatrix() const;

            // Función que devuelve un vector con los ángulos de Euler
            const dfv::Vector3    GetOriEuler() const;
```

```
56          private:
57              ros::NodeHandle&  node_handle;
58              unsigned int      mt_index;
59
60              std::string       acc_topic_name;
61              std::string       gyr_topic_name;
62              std::string       mag_topic_name;
63              std::string       ori_quat_topic_name;
64              std::string       ori_matrix_topic_name;
65              std::string       ori_euler_topic_name;
66
67              CmtOutputMode     output_mode;
68              CmtOutputSettings output_settings;
69
70              dfv::Vector3      acc;
71              dfv::Vector3      gyr;
72              dfv::Vector3      mag;
73
74              dfv::Quaternion   ori_quat;
75              dfv::Matrix       ori_matrix;
76              dfv::Vector3      ori_euler;
77
78              dfv::Vector3      position_lla;
79              double            temperature;
80
81              ros::Subscriber   acc_subscriber;
82              ros::Subscriber   gyr_subscriber;
83              ros::Subscriber   mag_subscriber;
84              ros::Subscriber   ori_quat_subscriber;
85              ros::Subscriber   ori_matrix_subscriber;
86              ros::Subscriber   ori_euler_subscriber;
87
88              void              AccSubCallback(const geometry_msgs::Vector3Stamped::
                    ConstPtr& msg);
89              void              GyrSubCallback(const geometry_msgs::Vector3Stamped::
                    ConstPtr& msg);
90              void              MagSubCallback(const geometry_msgs::Vector3Stamped::
                    ConstPtr& msg);
91              void              OriQuatSubCallback(const geometry_msgs::QuaternionStamped
                    ::ConstPtr& msg);
92              void              OriMatrixSubCallback(const std_msgs::Float64MultiArray::
                    ConstPtr& msg);
93              void              OriEulerSubCallback(const std_msgs::Float64MultiArray::
                    ConstPtr& msg);
94      };
95
96      class SensorSubscriberList
97      {
98          public:
99              SensorSubscriberList(ros::NodeHandle& node_handle_);
100             ~SensorSubscriberList();
101
102             // Función que devuelve el número de sensores detectados
103             unsigned int GetMtCount() const;
104
105             // Función que devuelve el vector aceleración
106             const dfv::Vector3    GetAcc(unsigned int mt_index) const;
107
```

```
108             // Función que devuelve el vector giróscopo
109             const dfv::Vector3    GetGyr(unsigned int mt_index) const;
110 
111             // Función que devuelve el vector campo magnético
112             const dfv::Vector3    GetMag(unsigned int mt_index) const;
113 
114             // Función que devuelve el cuaternión de orientación
115             const dfv::Quaternion GetOriQuat(unsigned int mt_index) const;
116 
117             // Función que devuelve la matriz de orientación
118             const dfv::Matrix     GetOriMatrix(unsigned int mt_index) const;
119 
120             // Función que devuelve un vector con los ángulos de Euler
121             const dfv::Vector3    GetOriEuler(unsigned int mt_index) const;
122 
123         private:
124             ros::NodeHandle node_handle;
125             unsigned int mt_count;
126             SensorSubscriber** sensors;
127 
128     };
129 }
130 
131 #endif
```

1.7. xsens_sensor_subscriber.cpp

```
#include <xsens_driver/xsens_sensor_subscriber.h>

namespace xsens
{
    SensorSubscriber::SensorSubscriber(unsigned int mt_index_, ros::NodeHandle&
        node_handle_):
        node_handle(node_handle_), mt_index(mt_index_)
    {
        std::stringstream ss;
        ss << "/xsens_node/sensor" << this->mt_index << "/acc";
        this->acc_topic_name = ss.str();

        ss.str(std::string());
        ss << "/xsens_node/sensor" << this->mt_index << "/gyr";
        this->gyr_topic_name = ss.str();

        ss.str(std::string());
        ss << "/xsens_node/sensor" << this->mt_index << "/mag";
        this->mag_topic_name = ss.str();

        ss.str(std::string());
        ss << "xsens_node/sensor" << this->mt_index << "/ori_quat";
        this->ori_quat_topic_name = ss.str();

        ss.str(std::string());
        ss << "/xsens_node/sensor" << this->mt_index << "/ori_matrix";
        this->ori_matrix_topic_name = ss.str();

        ss.str(std::string());
        ss << "/xsens_node/sensor" << this->mt_index << "/ori_euler";
        this->ori_euler_topic_name = ss.str();

        int param;
        this->node_handle.param<int>("/xsens_node/output_mode", param, 0);
        this->output_mode = param;
        this->node_handle.param<int>("/xsens_node/output_settings", param, 0);
        this->output_settings = param;

        this->SubscribeToTopics();

    }

    SensorSubscriber::~SensorSubscriber()
    {

    }

    bool SensorSubscriber::SubscribeToTopics()
    {
        if((this->output_mode & CMT_OUTPUTMODE_CALIB) != 0)
        {
            ROS_INFO("[SensorSubscriber] Subscribing to calibrated data topics...");
            this->acc_subscriber = this->node_handle.subscribe(this->acc_topic_name,
                                                               1,
```

```
                                                    &SensorSubscriber::
                                                        AccSubCallback,
                                                    this);
        this->gyr_subscriber = this->node_handle.subscribe(this->gyr_topic_name,
                                                    1,
                                                    &SensorSubscriber::
                                                        GyrSubCallback,
                                                    this);
        this->mag_subscriber = this->node_handle.subscribe(this->mag_topic_name,
                                                    1,
                                                    &SensorSubscriber::
                                                        MagSubCallback,
                                                    this);
    }

    if((this->output_mode & CMT_OUTPUTMODE_ORIENT) != 0)
    {
        if((this->output_settings & CMT_OUTPUTSETTINGS_ORIENTMODE_MASK) ==
            CMT_OUTPUTSETTINGS_ORIENTMODE_QUATERNION)
        {
            ROS_INFO("[SensorSubscriber] Subscribing to ori_quat topic...");
            this->ori_quat_subscriber = this->node_handle.subscribe(this->
                ori_quat_topic_name,
                                                        1,
                                                        &SensorSubscriber::
                                                            OriQuatSubCallback
                                                            ,
                                                        this);
        }

        if((this->output_settings & CMT_OUTPUTSETTINGS_ORIENTMODE_MASK) ==
            CMT_OUTPUTSETTINGS_ORIENTMODE_MATRIX)
        {
            ROS_INFO("[SensorSubscriber] Subscribing to ori_matrix topic...");
            this->ori_matrix_subscriber = this->node_handle.subscribe(this->
                ori_matrix_topic_name,
                                                         1,
                                                         &SensorSubscriber::
                                                             OriMatrixSubCallback
                                                             ,
                                                         this);
        }

        if((this->output_settings & CMT_OUTPUTSETTINGS_ORIENTMODE_MASK) ==
            CMT_OUTPUTSETTINGS_ORIENTMODE_EULER)
        {
            ROS_INFO("[SensorSubscriber] Subscribing to ori_euler topic...");
            this->ori_euler_subscriber = this->node_handle.subscribe(this->
                ori_euler_topic_name,
                                                        1,
                                                        &SensorSubscriber::
                                                            OriEulerSubCallback
                                                            ,
                                                        this);
        }
    }

    return true;
```

```
97      }
98
99      const dfv::Vector3 SensorSubscriber::GetAcc() const
100     {
101         return dfv::Vector3(this->acc);
102     }
103
104     const dfv::Vector3 SensorSubscriber::GetGyr() const
105     {
106         return dfv::Vector3(this->gyr);
107     }
108
109     const dfv::Vector3 SensorSubscriber::GetMag() const
110     {
111         return dfv::Vector3(this->mag);
112     }
113
114     const dfv::Quaternion SensorSubscriber::GetOriQuat() const
115     {
116         return dfv::Quaternion(this->ori_quat);
117     }
118
119     const dfv::Matrix SensorSubscriber::GetOriMatrix() const
120     {
121         return dfv::Matrix(this->ori_matrix);
122     }
123
124     const dfv::Vector3 SensorSubscriber::GetOriEuler() const
125     {
126         return dfv::Vector3(this->ori_euler);
127     }
128
129     void SensorSubscriber::AccSubCallback(const geometry_msgs::Vector3Stamped::ConstPtr
            & msg)
130     {
131         this->acc = dfv::Vector3(msg->vector.x, msg->vector.y, msg->vector.z);
132     }
133
134     void SensorSubscriber::GyrSubCallback(const geometry_msgs::Vector3Stamped::ConstPtr
            & msg)
135     {
136         this->gyr = dfv::Vector3(msg->vector.x, msg->vector.y, msg->vector.z);
137     }
138
139     void SensorSubscriber::MagSubCallback(const geometry_msgs::Vector3Stamped::ConstPtr
            & msg)
140     {
141         this->mag = dfv::Vector3(msg->vector.x, msg->vector.y, msg->vector.z);
142     }
143
144     void SensorSubscriber::OriQuatSubCallback(const geometry_msgs::QuaternionStamped::
            ConstPtr& msg)
145     {
146         this->ori_quat = dfv::Quaternion(msg->quaternion.w, msg->quaternion.x, msg->
                quaternion.y, msg->quaternion.z);
147     }
148
```

```
149     void SensorSubscriber::OriMatrixSubCallback(const std_msgs::Float64MultiArray::
            ConstPtr& msg)
150     {
151         dfv::Matrix m(3);
152         m.Set(0, 0 , msg->data[0]);
153         m.Set(0, 1 , msg->data[1]);
154         m.Set(0, 2 , msg->data[2]);
155         m.Set(1, 0 , msg->data[3]);
156         m.Set(1, 1 , msg->data[4]);
157         m.Set(1, 2 , msg->data[5]);
158         m.Set(2, 0 , msg->data[6]);
159         m.Set(2, 1 , msg->data[7]);
160         m.Set(2, 2 , msg->data[8]);
161         this->ori_matrix = m;
162     }
163 
164     void SensorSubscriber::OriEulerSubCallback(const std_msgs::Float64MultiArray::
            ConstPtr& msg)
165     {
166         this->ori_euler = dfv::Vector3(msg->data[0], msg->data[1], msg->data[2]);
167     }
168 
169 // ========================================== //
170 //         Clase SensorSubscriberList       //
171 // ========================================== //
172 
173     SensorSubscriberList::SensorSubscriberList(ros::NodeHandle& node_handle_):
174         node_handle(node_handle_)
175     {
176         int param;
177         this->node_handle.param<int>("/xsens_node/sensor_count", param, 0);
178         this->mt_count = param;
179         this->sensors = new SensorSubscriber*[this->mt_count];
180 
181         for(int i = 0; i < param; ++i)
182         {
183             this->sensors[i] = new SensorSubscriber(i, this->node_handle);
184         }
185     }
186 
187     SensorSubscriberList::~SensorSubscriberList()
188     {
189         for(int i = 0; i < this->mt_count; ++i)
190         {
191             delete this->sensors[i];
192         }
193         delete this->sensors;
194     }
195 
196     unsigned int SensorSubscriberList::GetMtCount() const
197     {
198         return this->mt_count;
199     }
200 
201     const dfv::Vector3 SensorSubscriberList::GetAcc(unsigned int mt_index) const
202     {
203         return dfv::Vector3(this->sensors[mt_index]->GetAcc());
204     }
```

```
205
206     const dfv::Vector3 SensorSubscriberList::GetGyr(unsigned int mt_index) const
207     {
208         return dfv::Vector3(this->sensors[mt_index]->GetGyr());
209     }
210
211     const dfv::Vector3 SensorSubscriberList::GetMag(unsigned int mt_index) const
212     {
213         return dfv::Vector3(this->sensors[mt_index]->GetMag());
214     }
215
216     const dfv::Quaternion SensorSubscriberList::GetOriQuat(unsigned int mt_index) const
217     {
218         return dfv::Quaternion(this->sensors[mt_index]->GetOriQuat());
219     }
220
221     const dfv::Matrix SensorSubscriberList::GetOriMatrix(unsigned int mt_index) const
222     {
223         return dfv::Matrix(this->sensors[mt_index]->GetOriMatrix());
224     }
225
226     const dfv::Vector3 SensorSubscriberList::GetOriEuler(unsigned int mt_index) const
227     {
228         return dfv::Vector3(this->sensors[mt_index]->GetOriEuler());
229     }
230
231 }
```

1.8. utils.h

```
#ifndef XSENS_DRIVER_UTILS_H
#define XSENS_DRIVER_UTILS_H

#include <xsens_driver/cmtdef.h>
#include <dfv/dfv.h>

namespace xsens
{
    const CmtMatrix     DfvToCmtMatrix(const dfv::Matrix& m);
    const dfv::Matrix   CmtToDfvMatrix(const CmtMatrix& m);
    const CmtVector     DfvToCmtVector(const dfv::Vector3& v);
    const dfv::Vector3  CmtToDfvVector(const CmtVector& v);
    const CmtShortVector DfvToCmtShortVector(const dfv::Vector3& v);
    const dfv::Vector3  CmtToDfvShortVector(const CmtShortVector& v);


    const geometry_msgs::Vector3 ToVector3Msg(const dfv::Vector3& v);
    const geometry_msgs::Vector3Stamped ToVector3StampedMsg(const dfv::Vector3& v);
}

#endif
```

1.9. utils.cpp

```cpp
#include <xsens_driver/utils.h>

namespace xsens
{
    const CmtMatrix DfvToCmtMatrix(const dfv::Matrix& m)
    {
        CmtMatrix result;
        if(m.GetRows() == 3 && m.GetColumns() == 3)
        {
            for(unsigned int j = 0; j < m.GetRows(); ++j)
            {
                for(unsigned int i = 0; i < m.GetColumns(); ++i)
                {
                    result.m_data[j][i] = m.Get(j, i);
                }
            }
        }
        return result;
    }

    const dfv::Matrix CmtToDfvMatrix(const CmtMatrix& m)
    {
        dfv::Matrix result(3, 3);
        for(unsigned int j = 0; j < 3; ++j)
        {
            for(unsigned int i = 0; i < 3; ++i)
            {
                result.Set(j, i, m.m_data[j][i]);
            }
        }
        return result;
    }

    const CmtVector DfvToCmtVector(const dfv::Vector3& v)
    {
        CmtVector result;
        result.m_data[0] = v.x;
        result.m_data[1] = v.y;
        result.m_data[2] = v.z;
        return result;
    }

    const dfv::Vector3 CmtToDfvShortVector(const CmtShortVector& v)
    {
        return dfv::Vector3(v.m_data[0], v.m_data[1], v.m_data[2]);
    }

    const CmtShortVector DfvToCmtShortVector(const dfv::Vector3& v)
    {
        CmtShortVector result;
        result.m_data[0] = v.x;
        result.m_data[1] = v.y;
        result.m_data[2] = v.z;
        return result;
    }
```

```
56
57      const dfv::Vector3 CmtToDfvVector(const CmtVector& v)
58      {
59          return dfv::Vector3(v.m_data[0], v.m_data[1], v.m_data[2]);
60      }
61
62      const geometry_msgs::Vector3 ToVector3Msg(const dfv::Vector3& v)
63      {
64          geometry_msgs::Vector3 msg;
65          msg.x = v.x;
66          msg.y = v.y;
67          msg.z = v.z;
68          return msg;
69      }
70
71      const geometry_msgs::Vector3Stamped ToVector3StampedMsg(const dfv::Vector3& v)
72      {
73          geometry_msgs::Vector3Stamped msg;
74          msg.header.stamp = ros::Time::now();
75          msg.vector.x = v.x;
76          msg.vector.y = v.y;
77          msg.vector.z = v.z;
78          return msg;
79      }
80  }
```

Capítulo 2

PAQUETE dfv

2.1. quaternion.h

```
/* Clase Quaternion
 * Incluye operaciones para creación y manipulación
 * de cuaterniones de orientación
 *
 * Autor: Daniel Fernández Villanueva
 */

#ifndef Quaternion_H
#define Quaternion_H

#include <iostream>
#include <cmath>
#include <sstream>
#include <dfv/vector3.h>
#include <dfv/utils.h>
#include <tf/transform_datatypes.h>


namespace dfv
{
    class Vector3;

    class Quaternion
    {
        public:

            Quaternion();
            Quaternion(double w_, double x_, double y_, double z_);
            explicit Quaternion(const Vector3& v);
            ~Quaternion();

            // ******** Operador de asignación ******** //
            Quaternion& operator=(const Quaternion& q);

            // ******** Operadores de asignación compuestos ******** //
            Quaternion& operator+=(const Quaternion& q);
            Quaternion& operator-=(const Quaternion& q);
            Quaternion& operator*=(const double k);

            // ******** Operadores aritméticos binarios ******** //
```

```
41          const Quaternion operator+(const Quaternion& q) const;
42          const Quaternion operator-(const Quaternion& q) const;
43          friend const Quaternion operator*(double k, Quaternion& q);
44          const Quaternion operator*(double k) const;
45          // Producto de Hamilton:
46          const Quaternion operator*(const Quaternion& q) const;
47
48          // ******** Operadores de comparación ******** //
49          bool operator==(const Quaternion& q) const;
50          bool operator!=(const Quaternion& q) const;
51
52          // Función que devuelve una representación
53          // del cuaternión como texto:
54          std::string          ToString() const;
55
56          // Función que devuelve el módulo del cuaternión:
57          double               GetModulus() const;
58
59          // Función que normaliza el cuaternión:
60          void                 Normalize();
61
62          // Función que devuelve el conjugado del cuaternión:
63          const Quaternion     GetConjugate() const;
64
65          // Función que devuelve el cuaternión de rotación
66          // definido por el eje [axisx_, axisy_, axisz_]
67          // y el ángulo angle_:
68          static const Quaternion GetRotationQuaternion(const double axisx_,
69                                                        const double axisy_,
70                                                        const double axisz_,
71                                                        const double angle_);
72
73          // Función que devuelve el cuaternión de rotación
74          // definido por el eje axis_
75          // y el ángulo angle_:
76          static const Quaternion GetRotationQuaternion(const Vector3& axis_,
77                                                        const double angle_);
78
79          // Función que devuelve el cuaternión de rotación
80          // definido por el eje perpendicular a v_before y v_after:
81          static const Quaternion GetRotationQuaternion(const Vector3& v_before,
82                                                        const Vector3& v_after);
83
84          // Función que devuelve el cuaternión de rotación
85          // del conjunto de vectores v1 y v2
86          // Se supone que v1 y v2 son perpendiculares entre sí:
87          static const Quaternion GetRotationQuaternion(const Vector3& v1_before,
88                                                        const Vector3& v1_after,
89                                                        const Vector3& v2_before,
90                                                        const Vector3& v2_after);
91
92
93          // Función que descompone el cuaternión en otros tres cuaterniones
94          // definidos por los ejes v1, v2 y v3, y unos valores iniciales
95          // de los ángulos angle_1, angle_2 y angle_3
96          // Los valores finales de angle_1, angle_2 y angle_3 son
97          // los resultados obtenidos para los ángulos asociados
98          // a los cuaterniones.
```

```
99            // Los vectores v1, v2 y v3 tienen que ser linearmente
100           // independientes para asegurar la obtención de un resultado
101           // correcto.
102           void                  Decompose(double& angle_1,
103                                           double& angle_2,
104                                           double& angle_3,
105                                           const Vector3& v1,
106                                           const Vector3& v2,
107                                           const Vector3& v3) const;
108
109           // Función para obtener el eje y el ángulo asociados al
110           // cuaternión:
111           void                  GetAxisAndAngle(Vector3& vector,
112                                                 double& angle) const;
113
114           // Función para obtener el roll, pitch y yaw del cuaternión
115           void                  GetRPY(double& roll, double& pitch, double& yaw,
              unsigned int solution = 1);
116
117           // Función que devuelve el cuaternión de diferencia entre dos cuaterniones
118           static const Quaternion GetDifference(const Quaternion& q1, const
              Quaternion& q2);
119
120           // Quaterniones unitarios colineales a las componentes w, x, y, z:
121           static const Quaternion identity;
122           static const Quaternion i;
123           static const Quaternion j;
124           static const Quaternion k;
125
126           // Componentes del cuaternión:
127           double w;
128           double x;
129           double y;
130           double z;
131
132       protected:
133       private:
134
135
136   };
137
138   std::ostream& operator<<(std::ostream& os, const Quaternion& q);
139
140 }
141
142 #endif // Quaternion_H
```

2.2. quaternion.cpp

```
#include "dfv/quaternion.h"

namespace dfv
{

    Quaternion::Quaternion(): w(0), x(0), y(0), z(0)
    {
        //ctor
    }

    Quaternion::Quaternion(double w_, double x_, double y_, double z_): w(w_), x(x_), y
        (y_), z(z_)
    {

    }

    Quaternion::Quaternion(const Vector3& v):
        w(0.0), x(v.x), y(v.y), z(v.z)
    {

    }

    Quaternion::~Quaternion()
    {
        //dtor
    }

    Quaternion& Quaternion::operator=(const Quaternion& q)
    {
        if(this != &q)
        {
            this->w = q.w;
            this->x = q.x;
            this->y = q.y;
            this->z = q.z;
        }

        return *this;
    }

    Quaternion& Quaternion::operator+=(const Quaternion& q)
    {
        this->w += q.w;
        this->x += q.x;
        this->y += q.y;
        this->z += q.z;

        return *this;
    }

    Quaternion& Quaternion::operator-=(const Quaternion& q)
    {
        this->w -= q.w;
        this->x -= q.x;
        this->y -= q.y;
```

```
55          this->z -= q.z;
56
57          return *this;
58      }
59
60      Quaternion& Quaternion::operator*=(const double k)
61      {
62          this->w *= k;
63          this->x *= k;
64          this->y *= k;
65          this->z *= k;
66
67          return *this;
68      }
69
70      const Quaternion Quaternion::operator+(const Quaternion& q) const
71      {
72          return Quaternion(*this) += q;
73      }
74
75      const Quaternion Quaternion::operator-(const Quaternion& q) const
76      {
77          return Quaternion(*this) -= q;
78      }
79
80      const Quaternion operator*(double k, Quaternion& q)
81      {
82          return Quaternion(q) *= k;
83      }
84
85      const Quaternion Quaternion::operator*(double k) const
86      {
87          return Quaternion(*this) *= k;
88      }
89
90      const Quaternion Quaternion::operator*(const Quaternion& q) const
91      {
92          return Quaternion(this->w*q.w - this->x*q.x - this->y*q.y - this->z*q.z,
93                            this->x*q.w + this->w*q.x - this->z*q.y + this->y*q.z,
94                            this->y*q.w + this->z*q.x + this->w*q.y - this->x*q.z,
95                            this->z*q.w - this->y*q.x + this->x*q.y + this->w*q.z);
96      }
97
98      bool Quaternion::operator==(const Quaternion& q) const
99      {
100         return (this->w == q.w) && (this->x == q.x) && (this->y == q.y) && (this->z ==
                q.z);
101     }
102
103     bool Quaternion::operator!=(const Quaternion& q) const
104     {
105         return !(*this == q);
106     }
107
108     std::string Quaternion::ToString() const
109     {
110         std::stringstream ss;
```

```
111         ss << this->w << " + " << this->x << "*i + " << this->y << "*j + " << this->z
                << "*k";
112         return ss.str();
113     }
114
115     double Quaternion::GetModulus() const
116     {
117         return sqrt(pow(this->w, 2) + pow(this->x, 2) + pow(this->y, 2) + pow(this->z,
                2));
118     }
119
120     void Quaternion::Normalize()
121     {
122         double l = this->GetModulus();
123         if(l == 0.f)
124         {
125             this->w = 0.f;
126             this->x = 0.f;
127             this->y = 0.f;
128             this->z = 0.f;
129         }
130         else
131         {
132             this->w /= l;
133             this->x /= l;
134             this->y /= l;
135             this->z /= l;
136         }
137     }
138
139     const Quaternion Quaternion::GetConjugate() const
140     {
141         return Quaternion(this->w, -this->x, -this->y, -this->z);
142     }
143
144     const Quaternion Quaternion::GetRotationQuaternion(const double axisx_,
145                                                        const double axisy_,
146                                                        const double axisz_,
147                                                        const double angle_)
148     {
149         double l = sqrt(axisx_*axisx_ + axisy_*axisy_ + axisz_*axisz_);
150         return Quaternion(cos(angle_ / 2.0),
151                           axisx_/l * sin(angle_ / 2.0),
152                           axisy_/l * sin(angle_ / 2.0),
153                           axisz_/l * sin(angle_ / 2.0));
154     }
155
156     const Quaternion Quaternion::GetRotationQuaternion(const Vector3& axis_,
157                                                        const double angle_)
158     {
159         double l = axis_.GetMagnitude();
160         return Quaternion(cos(angle_ / 2.0),
161                           axis_.x/l * sin(angle_ / 2.0),
162                           axis_.y/l * sin(angle_ / 2.0),
163                           axis_.z/l * sin(angle_ / 2.0));
164     }
165
166     const Quaternion Quaternion::GetRotationQuaternion(const Vector3& v_before,
```

```
                                                const Vector3& v_after)
{
    Vector3 vu = v_before.GetNormalized();
    Vector3 vvu = v_after.GetNormalized();
    if(vu == vvu)
    {
        return Quaternion(1.0, 0.0, 0.0, 0.0);
    }
    Vector3 r = (vu ^ vvu).GetNormalized();
    if(r == Vector3(0.0, 0.0, 0.0))
    {
        r = ((vu ^ Vector3::i) + (vu ^ Vector3::j)).GetNormalized();
        return Quaternion::GetRotationQuaternion(r, pi);
    }
    else
    {
        double sin_theta = (vu ^ vvu).GetMagnitude();
        double cos_theta = vu * vvu;
        double theta = atan2(sin_theta, cos_theta);
        return Quaternion::GetRotationQuaternion(r, theta);
    }

}

const Quaternion Quaternion::GetRotationQuaternion(const Vector3& v1_before,
                                                const Vector3& v1_after,
                                                const Vector3& v2_before,
                                                const Vector3& v2_after)
{
    Quaternion q1 = Quaternion::GetRotationQuaternion(v1_before, v1_after);
    Vector3 p2(v2_before);
    Vector3 pp2 = p2.GetRotated(q1);
    Vector3 vv2(pp2);
    Vector3 r2 = v1_after;

    if((vv2 ^ v2_after)*v1_after > 0)
    {
        double sin_theta = (vv2 ^ v2_after).GetMagnitude();
        double cos_theta = vv2*v2_after;
        double theta = atan2(sin_theta, cos_theta);
        Quaternion q2 = Quaternion::GetRotationQuaternion(r2, theta);
        return q2*q1;
    }
    else
    {
        double sin_theta = -((vv2 ^ v2_after).GetMagnitude());
        double cos_theta = vv2*v2_after;
        double theta = atan2(sin_theta, cos_theta);
        Quaternion q2 = Quaternion::GetRotationQuaternion(r2, theta);
        return q2*q1;
    }
}

void Quaternion::Decompose(double& angle_1,
                           double& angle_2,
                           double& angle_3,
                           const Vector3& v1,
                           const Vector3& v2,
```

```
225                             const Vector3& v3) const
226     {
227         Quaternion q1 = Quaternion::GetRotationQuaternion(v1, angle_1);
228         Quaternion q2 = Quaternion::GetRotationQuaternion(v2, angle_2);
229         Quaternion q3 = Quaternion::GetRotationQuaternion(v3, angle_3);
230         Quaternion qt = q3*q2*q1;
231 
232         const double err = 0.0001;
233         const double delta_angle = 0.0001;
234         const unsigned int tries = 10000;
235         unsigned int k = 0;
236 
237         Quaternion qr = *this;
238 
239         double dist = (qr - qt).GetModulus();
240 
241         while((dist >= err) && (k < tries))
242         {
243             for(int c1 = -1; c1 <= 1; ++c1)
244             {
245                 for(int c2 = -1; c2 <= 1; ++c2)
246                 {
247                     for (int c3 = -1; c3 <= 1; ++c3)
248                     {
249                         double new_angle_1 = angle_1 + delta_angle*c1;
250                         double new_angle_2 = angle_2 + delta_angle*c2;
251                         double new_angle_3 = angle_3 + delta_angle*c3;
252 
253                         Quaternion new_q1 = Quaternion::GetRotationQuaternion(v1,
                                new_angle_1);
254                         Quaternion new_q2 = Quaternion::GetRotationQuaternion(v2,
                                new_angle_2);
255                         Quaternion new_q3 = Quaternion::GetRotationQuaternion(v3,
                                new_angle_3);
256                         Quaternion new_qt = new_q3*new_q2*new_q1;
257 
258                         double new_dist = (qr - new_qt).GetModulus();
259 
260                         if(new_dist < dist)
261                         {
262                             angle_1 = new_angle_1;
263                             angle_2 = new_angle_2;
264                             angle_3 = new_angle_3;
265                             dist = new_dist;
266                         }
267                     }
268                 }
269             }
270             ++k;
271         }
272     }
273 
274     void Quaternion::GetAxisAndAngle(Vector3& vector, double& angle) const
275     {
276         Quaternion qr = *this;
277         Vector3 e(qr);
278         e.Normalize();
279         double cc = this->w;
```

```
280         double ss = ((qr - Quaternion(this->w, 0, 0, 0)) * Quaternion(e).GetConjugate()
                ).w;
281         double ang = 2*atan2(ss, cc);
282 
283         vector = e;
284         angle = ang;
285     }
286 
287     void Quaternion::GetRPY(double& roll, double& pitch, double& yaw, unsigned int
            solution)
288     {
289         tf::Quaternion tf_q(this->x, this->y, this->z, this->w);
290         tf::Matrix3x3(tf_q).getRPY(roll, pitch, yaw, solution);
291     }
292 
293     const Quaternion Quaternion::GetDifference(const Quaternion& q1, const Quaternion&
            q2)
294     {
295         return (q1.GetConjugate())*q2;
296     }
297 
298     const Quaternion Quaternion::identity = Quaternion(1,0,0,0);
299     const Quaternion Quaternion::i     = Quaternion(0,1,0,0);
300     const Quaternion Quaternion::j     = Quaternion(0,0,1,0);
301     const Quaternion Quaternion::k     = Quaternion(0,0,0,1);
302 
303     std::ostream& operator<<(std::ostream& os, const Quaternion& q)
304     {
305         return os << q.ToString();
306     }
307 
308 }
```

2.3. vector3.h

```
/* Clase Vector3
 * Incluye operaciones para creación y manipulación
 * de vectores de 3 dimensiones
 *
 * Autor: Daniel Fernández Villanueva
 */

#ifndef VECTOR3_H
#define VECTOR3_H

#include <string>
#include <sstream>
#include <cmath>
#include <dfv/quaternion.h>
#include <ros/ros.h>
#include <geometry_msgs/Vector3.h>
#include <geometry_msgs/Vector3Stamped.h>

namespace dfv
{

    class Quaternion;

    class Vector3
    {
        public:
            Vector3();
            Vector3(double x_, double y_, double z_);
            explicit Vector3(const Quaternion& q);
            virtual ~Vector3();

            // ******** Operador de asignación ******** //
            Vector3& operator=(const Vector3& v);

            // ******** Operadores de asignación compuestos ******** //
            Vector3& operator+=(const Vector3& v);
            Vector3& operator-=(const Vector3& v);
            Vector3& operator*=(const double k);

            // ******** Operadores aritméticos binarios ******** //
            const Vector3 operator+(const Vector3& v) const;
            const Vector3 operator-(const Vector3& v) const;
            friend const Vector3 operator*(double k, Vector3& v);
            const Vector3 operator*(double k) const;
            // Producto escalar:
            double operator*(const Vector3& v) const;
            // Producto vectorial:
            const Vector3 operator^(const Vector3& v) const;

            // ******** Operadores de comparación ******** //
            bool operator==(const Vector3& v) const;
            bool operator!=(const Vector3& v) const;

            // Funcion que devuelve la magnitud del vector:
            double        GetMagnitude() const;
```

```
56
57              // Funcion que normaliza el vector:
58              void           Normalize();
59
60              // Funcion que devuelve el vector normalizado:
61              const Vector3  GetNormalized() const;
62
63              // Funcion que devuelve un vector de longitud
64              // k veces el original con la misma dirección
65              // y sentido:
66              const Vector3  GetScalated(double k) const;
67
68              // Función que rota el vector de acuerdo con el cuaternión
69              // pasado como parámetro:
70              Vector3&       Rotate(const Quaternion q);
71
72              // Devuelve el vector rotado según el cuaternión pasado
73              // como parámetro:
74              const Vector3  GetRotated(const Quaternion& q) const;
75
76              // Función que devuelve una representación
77              // del vector como texto:
78              std::string    ToString() const;
79
80              // Vector unitario codireccional al eje x:
81              static Vector3 i;
82
83              // Vector unitario codireccional al eje y:
84              static Vector3 j;
85
86              // Vector unitario codireccional al eje z:
87              static Vector3 k;
88
89              // Componentes del vector:
90              double x;
91              double y;
92              double z;
93
94          protected:
95          private:
96
97      };
98
99      std::ostream& operator<<(std::ostream& os, const Vector3& v);
100
101 }
102
103 #endif // Vector3_H
```

2.4. vector3.cpp

```
#include <dfv/vector3.h>

namespace dfv
{
    Vector3::Vector3():
        x(0), y(0), z(0)
    {

    }

    Vector3::Vector3(double x_, double y_, double z_):
        x(x_), y(y_), z(z_)
    {

    }

    Vector3::Vector3(const Quaternion& q):
        x(q.x), y(q.y), z(q.z)
    {

    }

    Vector3::~Vector3()
    {
        //dtor
    }

    Vector3& Vector3::operator=(const Vector3& v)
    {
        if(this != &v)
        {
            this->x = v.x;
            this->y = v.y;
            this->z = v.z;
        }

        return *this;
    }

    Vector3& Vector3::operator+=(const Vector3& v)
    {
        this->x += v.x;
        this->y += v.y;
        this->z += v.z;

        return *this;
    }

    Vector3& Vector3::operator-=(const Vector3& v)
    {
        this->x -= v.x;
        this->y -= v.y;
        this->z -= v.z;

        return *this;
```

```
56      }
57
58      Vector3& Vector3::operator*=(const double k)
59      {
60          this->x *= k;
61          this->y *= k;
62          this->z *= k;
63
64          return *this;
65      }
66
67      const Vector3 Vector3::operator+(const Vector3& v) const
68      {
69          return Vector3(*this) += v;
70      }
71
72      const Vector3 Vector3::operator-(const Vector3& v) const
73      {
74          return Vector3(*this) -= v;
75      }
76
77      const Vector3 operator*(double k, Vector3& v)
78      {
79          return Vector3(v) *= k;
80      }
81
82      const Vector3 Vector3::operator*(double k) const
83      {
84          return Vector3(*this) *= k;
85      }
86
87      double Vector3::operator*(const Vector3& v) const
88      {
89          return this->x * v.x + this->y * v.y + this->z * v.z;
90      }
91
92      const Vector3 Vector3::operator^(const Vector3& v) const
93      {
94          return Vector3(this->y*v.z - this->z*v.y,
95                         this->z*v.x - this->x*v.z,
96                         this->x*v.y - this->y*v.x);
97      }
98
99      bool Vector3::operator==(const Vector3& v) const
100     {
101         return (this->x == v.x) && (this->y == v.y) && (this->z == v.z);
102     }
103
104     bool Vector3::operator!=(const Vector3& v) const
105     {
106         return !(*this == v);
107     }
108
109     double Vector3::GetMagnitude() const
110     {
111         return sqrt(pow(this->x, 2.0) + pow(this->y, 2.0) + pow(this->z, 2.0));
112     }
113
```

```
114     void Vector3::Normalize()
115     {
116         double mag = this->GetMagnitude();
117         if(mag > 0)
118         {
119             this->x /= mag;
120             this->y /= mag;
121             this->z /= mag;
122         }
123         else
124         {
125             this->x = 0.0;
126             this->y = 0.0;
127             this->z = 0.0;
128         }
129 
130     }
131 
132     const Vector3 Vector3::GetNormalized() const
133     {
134         double mag = this->GetMagnitude();
135 
136         if(mag > 0)
137         {
138             Vector3 result;
139             result.x = this->x / mag;
140             result.y = this->y / mag;
141             result.z = this->z / mag;
142             return result;
143         }
144         else
145         {
146             return Vector3(0.0, 0.0, 0.0);
147         }
148     }
149 
150     const Vector3 Vector3::GetScalated(double k) const
151     {
152         return Vector3(this->x*k, this->y*k, this->z*k);
153     }
154 
155     Vector3& Vector3::Rotate(const Quaternion q)
156     {
157         *this = this->GetRotated(q);
158         return *this;
159     }
160 
161     const Vector3 Vector3::GetRotated(const Quaternion& q) const
162     {
163         Quaternion v(*this);
164         return Vector3(q*v*(q.GetConjugate()));
165     }
166 
167     std::string Vector3::ToString() const
168     {
169         std::stringstream ss;
170         ss << "[" << this->x << ", " << this->y << ", " << this->z << "]";
171         return ss.str();
```

```
172     }
173
174     Vector3 Vector3::i = Vector3(1, 0, 0);
175     Vector3 Vector3::j = Vector3(0, 1, 0);
176     Vector3 Vector3::k = Vector3(0, 0, 1);
177
178     std::ostream& operator<<(std::ostream& os, const Vector3& v)
179     {
180         return os << v.ToString();
181     }
182 }
```

2.5. matrix.h

```
/* Clase Matrix
 * Incluye operaciones para creación y manipulación
 * de matrices de cualquier dimensión
 *
 * Autor: Daniel Fernández Villanueva
 */

#ifndef MATRIX_H
#define MATRIX_H

#include <iostream>
#include <vector>
#include <sstream>
#include <cstdlib>

namespace dfv
{

    class Matrix
    {
        public:
            Matrix();
            Matrix(unsigned int size);
            Matrix(unsigned int rows, unsigned int columns);
            virtual ~Matrix();

            // ******** Operador de asignación ******** //
            Matrix& operator=(const Matrix& q);

            // ******** Operadores de asignación compuestos ******** //
            Matrix& operator+=(const Matrix& q);
            Matrix& operator-=(const Matrix& q);
            Matrix& operator*=(const double k);

            // ******** Operadores aritméticos binarios ******** //
            const Matrix operator+(const Matrix& q) const;
            const Matrix operator-(const Matrix& q) const;
            friend const Matrix operator*(double k, Matrix& q);
            const Matrix operator*(double k) const;
            // Producto de Matrices:
            const Matrix operator*(const Matrix& q) const;

            // ******** Operadores de comparación ******** //
            bool operator==(const Matrix& q) const;
            bool operator!=(const Matrix& q) const;

            // Función que devuelve una representación
            // de la matriz como texto:
            std::string          ToString() const;

            // Función que crea una matriz de rows filas y columns columnas
            Matrix&           Create(unsigned int rows, unsigned int columns, double
                value = 0);

            // Función que devuelve el elemento situado en la fila row y columna col
```

```
55            double            Get(unsigned int row, unsigned int col) const;
56
57            // Función que permie cambiar el valor del elemento situado en la fila row
                  y columna col
58            void              Set(unsigned int row, unsigned int col, double value);
59
60            // Función que devuelve el número de filas de la matriz
61            unsigned int      GetRows() const;
62
63            // Función que devuelve el número de columnas de la matriz
64            unsigned int      GetColumns() const;
65
66            // Función que devuleve el menor asociado a la fila row y columna column
67            Matrix            GetMinor(unsigned int row, unsigned int column) const;
68
69            // Función que asigna un valor aleatoria ontre 0 y 1 a cada elemento de la
                  matriz
70            void              Randomize();
71
72            // Función que devuelve el determinante de la matriz
73            double            GetDeterminant() const;
74
75            // Función que devuelve la traspuesta de la matriz
76            const Matrix      GetTransposed() const;
77
78            // Función que devuelve el adjunto de la matriz
79            const Matrix      GetAdjoint() const;
80
81            // Función que devuelve la matriz transpuesta conjugada
82            const Matrix      GetAdjugate() const;
83
84            // Función que devuelve la inversa de la matriz
85            const Matrix      GetInverse() const;
86
87            // Operador equivalente a la función Get()
88            double            operator()(unsigned int row, unsigned int column) const;
89
90            // Función que devuelve una matriz unitaria con la dimensión dada
91            static const Matrix Identity(unsigned int size);
92
93        protected:
94        private:
95            unsigned int      rows;
96            unsigned int      columns;
97            std::vector<double> data;
98    };
99
100   std::ostream& operator<<(std::ostream& os, Matrix& m);
101
102}
103
104#endif // Matrix_H
```

2.6. matrix.cpp

```
#include <dfv/matrix.h>

namespace dfv
{

    Matrix::Matrix(): rows(0), columns(0)
    {

    }

    Matrix::Matrix(unsigned int size)
    {
        this->Create(size, size);
    }

    Matrix::Matrix(unsigned int rows, unsigned int columns)
    {
        this->Create(rows, columns);
    }

    Matrix::~Matrix()
    {
        //dtor
    }

    Matrix& Matrix::operator=(const Matrix& m)
    {
        if(this != &m)
        {
            this->Create(m.GetRows(), m.GetColumns());

            for(unsigned int j = 0; j < this->GetRows(); j++)
            {
                for(unsigned int i = 0; i < this->GetColumns(); i++)
                {
                    this->Set(j, i, m.Get(j, i));
                }
            }
        }

        return *this;
    }

    Matrix& Matrix::operator+=(const Matrix& m)
    {
        if(m.GetRows() == this->GetRows() && m.GetColumns() == this->GetColumns())
        {
            for(unsigned int j = 0; j < this->GetRows(); j++)
            {
                for(unsigned int i = 0; i < this->GetColumns(); i++)
                {
                    this->Set(j, i, this->Get(j, i) + m.Get(j, i));
                }
            }

```

```
56              return *this;
57          }
58          else
59          {
60              return *this; // arrojar error
61          }
62      }
63
64      Matrix& Matrix::operator-=(const Matrix& m)
65      {
66          if(m.GetRows() == this->GetRows() && m.GetColumns() == this->GetColumns())
67          {
68              for(unsigned int j = 0; j < this->GetRows(); j++)
69              {
70                  for(unsigned int i = 0; i < this->GetColumns(); i++)
71                  {
72                      this->Set(j, i, this->Get(j, i) - m.Get(j, i));
73                  }
74              }
75
76              return *this;
77          }
78          else
79          {
80              return *this; // arrojar error
81          }
82      }
83
84      Matrix& Matrix::operator*=(const double k)
85      {
86          for(unsigned int j = 0; j < this->GetRows(); j++)
87          {
88              for(unsigned int i = 0; i < this->GetColumns(); i++)
89              {
90                  this->Set(j, i, this->Get(j, i) * k);
91              }
92          }
93
94          return *this;
95      }
96
97      const Matrix Matrix::operator+(const Matrix& m) const
98      {
99          return Matrix(*this) += m;
100     }
101
102     const Matrix Matrix::operator-(const Matrix& m) const
103     {
104         return Matrix(*this) -= m;
105     }
106
107     const Matrix operator*(double k, Matrix& m)
108     {
109         return Matrix(m) *= k;
110     }
111
112     const Matrix Matrix::operator*(double k) const
113     {
```

```
114         return Matrix(*this) *= k;
115     }
116 
117     const Matrix Matrix::operator*(const Matrix& m) const
118     {
119         Matrix result;
120         if(this->GetColumns() == m.GetRows())
121         {
122             result.Create(this->GetRows(), m.GetColumns());
123             for(unsigned int j = 0; j < result.GetRows(); j++)
124             {
125                 for(unsigned int i = 0; i < result.GetColumns(); i++)
126                 {
127                     for(unsigned int k = 0; k < this->GetColumns(); k++)
128                     {
129                         result.Set(j, i, result.Get(j, i) + this->Get(j, k)*m.Get(k, i));
130                     }
131                 }
132             }
133         }
134         return result;
135     }
136 
137     bool Matrix::operator==(const Matrix& m) const
138     {
139         if(m.GetRows() == this->GetRows() && m.GetColumns() == this->GetColumns())
140         {
141             for(unsigned int j = 0; j < this->GetRows(); j++)
142             {
143                 for(unsigned int i = 0; i < this->GetColumns(); i++)
144                 {
145                     if(this->Get(j, i) != m.Get(j, i))
146                     {
147                         return false;
148                     }
149                 }
150             }
151 
152             return true;
153         }
154         else
155         {
156             return false;
157         }
158     }
159 
160     bool Matrix::operator!=(const Matrix& m) const
161     {
162         return !(*this == m);
163     }
164 
165     std::string Matrix::ToString() const
166     {
167         std::stringstream ss;
168 
169         for(unsigned int j = 0; j < this->GetRows(); j++)
170         {
171             for(unsigned int i = 0; i < this->GetColumns(); i++)
```

```
172             {
173                 if(this->Get(j, i) >= 0)
174                 {
175                     ss << " ";
176                 }
177                 ss << this->Get(j, i) << "\t";
178             }
179             ss << std::endl;
180         }
181 
182         return ss.str();
183     }
184 
185     Matrix& Matrix::Create(unsigned int rows, unsigned int columns, double value)
186     {
187         this->rows = rows;
188         this->columns = columns;
189         this->data.resize(rows*columns);
190         typename std::vector<double>::iterator it;
191         for(it = this->data.begin(); it != this->data.end(); it++)
192         {
193             *it = value;
194         }
195         return *this;
196     }
197 
198     double Matrix::Get(unsigned int row, unsigned int col) const
199     {
200         return this->data.at(row + col*this->columns);
201     }
202 
203     void Matrix::Set(unsigned int row, unsigned int col, double value)
204     {
205         this->data.at(row + col*this->columns) = value;
206     }
207 
208     unsigned int Matrix::GetRows() const
209     {
210         return this->rows;
211     }
212 
213     unsigned int Matrix::GetColumns() const
214     {
215         return this->columns;
216     }
217 
218     Matrix Matrix::GetMinor(unsigned int row, unsigned int column) const
219     {
220         Matrix result;
221         if(this->GetRows() == this->GetColumns())
222         {
223             result.Create(this->GetRows() - 1, this->GetColumns() - 1);
224             unsigned int row_count = 0;
225             unsigned int col_count = 0;
226             for(unsigned int j = 0; j < this->GetRows(); j++)
227             {
228                 if(j != row)
229                 {
```

```
                col_count = 0;
                for(unsigned int i = 0; i < this->GetColumns(); i++)
                {
                    if(i != column)
                    {
                        result.Set(row_count, col_count, this->Get(j, i));
                        col_count++;
                    }
                }
                row_count++;
            }
        }
    }
    return result;
}

void Matrix::Randomize()
{
    for(unsigned int j = 0; j < this->GetRows(); j++)
    {
        for(unsigned int i = 0; i < this->GetColumns(); i++)
        {
            this->Set(j, i, (double)rand() / (double)RAND_MAX);
        }
    }
}

double Matrix::GetDeterminant() const
{
    double result = 0;
    if(this->GetRows() == this->GetColumns())
    {
        if(this->GetRows() == 1)
        {
            return this->Get(0, 0);
        }
        else
        {
            for(unsigned int i = 0; i < this->GetColumns(); i++)
            {
                Matrix minor = this->GetMinor(0, i);
                result += (i % 2 == 0? 1.0 : -1.0)*this->Get(0, i)*minor.
                    GetDeterminant();
            }
        }
    }
    return result;
}

const Matrix Matrix::GetTransposed() const
{
    Matrix result;
    result.Create(this->GetColumns(), this->GetRows());
    for(unsigned int j = 0; j < this->GetRows(); j++)
    {
        for(unsigned int i = 0; i < this->GetColumns(); i++)
        {
            result.Set(j, i, this->Get(i, j));
```

```
287             }
288         }
289         return result;
290     }
291
292     const Matrix Matrix::GetAdjoint() const
293     {
294         Matrix result;
295         result.Create(this->GetRows(), this->GetColumns());
296         for(unsigned int j = 0; j < this->GetRows(); j++)
297         {
298             for(unsigned int i = 0; i < this->GetColumns(); i++)
299             {
300                 Matrix minor;
301                 minor = this->GetMinor(j, i);
302                 double temp = ((j+i)%2 == 0 ? 1.0 : -1.0) * minor.GetDeterminant();
303                 result.Set(j, i, temp);
304             }
305         }
306         return result;
307     }
308
309     const Matrix Matrix::GetAdjugate() const
310     {
311         Matrix result;
312         result.Create(this->GetRows(), this->GetColumns());
313         for(unsigned int j = 0; j < this->GetRows(); j++)
314         {
315             for(unsigned int i = 0; i < this->GetColumns(); i++)
316             {
317                 Matrix minor;
318                 minor = this->GetMinor(j, i);
319                 double temp = ((j+i)%2 == 0 ? 1.0 : -1.0) * minor.GetDeterminant();
320                 result.Set(i, j, temp);
321             }
322         }
323         return result;
324     }
325
326     const Matrix Matrix::GetInverse() const
327     {
328         Matrix result;
329         if((this->GetRows() == this->GetColumns()) && this->GetDeterminant() != 0.0)
330         {
331             Matrix temp = this->GetAdjugate();
332             temp *= (1.0 / this->GetDeterminant());
333             result = temp;
334         }
335         return result;
336     }
337
338     double Matrix::operator()(unsigned int row, unsigned int column) const
339     {
340         return this->Get(row, column);
341     }
342
343     const Matrix Matrix::Identity(unsigned int size)
344     {
```

```
345         Matrix result(size, size);
346 
347         for(int i = 0; i < size; ++i)
348         {
349             result.Set(i, i, 1.0);
350         }
351 
352         return result;
353     }
354 
355     std::ostream& operator<<(std::ostream& os, Matrix& m)
356     {
357         os << m.ToString();
358         return os;
359     }
360 
361 }
```

2.7. utils.h

```
#ifndef UTILS_H
#define UTILS_H

namespace dfv
{
    const long double pi =
        3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862280
        L;

    long double DegToRad(long double deg);
    long double RadToDeg(long double rad);
}

#endif
```

2.8. utils.cpp

```cpp
#include <dfv/utils.h>

namespace dfv
{

    long double DegToRad(long double deg)
    {
        return deg * pi / 180.0L;
    }

    long double RadToDeg(long double rad)
    {
        return rad * 180.0L / pi;
    }

}
```

2.9. dfv.h

```
#ifndef DFV_H
#define DFV_H

#include <dfv/utils.h>
#include <dfv/vector3.h>
#include <dfv/quaternion.h>
#include <dfv/matrix.h>

#endif
```

Capítulo 3

PAQUETE youbot controller

3.1. youbot_controller.h

```
/*
 * Programa que toma los datos de los topics
 * de tres sensores xsens, calcula los ángulos
 * de rotación entre cada sensor y los publica
 * en el topic para mover el brazo robótico del
 * robot Youbot.
 *
 * Autor: Daniel Fernández Villanueva
 * Mayo de 2013
 *
 */

#include <iostream>
#include <ros/ros.h>
#include <dfv/dfv.h>
#include <xsens_driver/xsens_sensor_subscriber.h>
#include <youbot_controller/youbot.h>

// Función que devuelve el equivalente al ángulo
// dentro del rango [0, 2*pi]
double NormalizeAngle(double angle);

// ángulos límite
const double ang_min[] = {0.010, 0.010, -5.026, 0.022, 0.110};
const double ang_max[] = {5.840, 2.617, -0.015, 3.429, 5.641};
const double sec_ang = 0.05;

std::vector<double> GetAngles(xsens::SensorSubscriberList& sensors);

int main(int argc, char** argv)
{
    ros::init(argc, argv, "youbot_controller");
    ros::NodeHandle node_handle;

    xsens::SensorSubscriberList sensor_subscriber_list(node_handle);
    unsigned int mt_count = sensor_subscriber_list.GetMtCount();

    // Si no hay sensores, salimos del programa
    if(mt_count == 0)
    {
```

```
41          ROS_ERROR("No MTs found. Quitting...");
42          return 1;
43      }
44      ROS_INFO("Detected IMU count: %d",mt_count);
45      if(mt_count != 3)
46      {
47          ROS_WARN("3 sensors are needed by this demo to work properly. "
48              "Please connect 3 Xsens IMUs to the Xbus Master and restart the program. "
49              "Quitting...");
50          return 1;
51      }
52
53      // Esperamos a que ROS llame a las funciones callback para
54      // leer los datos de los topics
55      ROS_INFO("Waiting for the topics to be read...");
56      while(sensor_subscriber_list.GetOriQuat(0) == dfv::Quaternion(0.0, 0.0, 0.0, 0.0))
57      {
58          ros::Duration(0.05).sleep();
59          ros::spinOnce();
60      }
61
62      Youbot youbot(node_handle);
63
64      std::vector<double> angs;
65
66      unsigned int c = 200;
67      double ang_0_offset = 0;
68      ROS_INFO("Calibrating Sensor Orientation...");
69
70      while(ros::ok() && c != 0)
71      {
72          angs = GetAngles(sensor_subscriber_list);
73          ang_0_offset = (ang_min[0] + ang_max[0]) / 2.0 - angs[0];
74          ros::Duration(0.05).sleep();
75          c--;
76      }
77      ROS_INFO("Calibration Done.");
78
79      // bucle principal
80      while(ros::ok())
81      {
82          // Obtención de los ángulos a partir de los cuaterniones de los sensores
83          angs = GetAngles(sensor_subscriber_list);
84
85          angs[0] += ang_0_offset;
86          angs[0] = NormalizeAngle(angs[0]);
87
88          // Corrección de los ángulos para que no superen los límites
89          for(int i = 0; i < 5; i++)
90          {
91              youbot.joint_positions[i] = (angs[i] < ang_min[i] + sec_ang) ?
92                  ang_min[i] + sec_ang : ((angs[i] > ang_max[i] - sec_ang) ?
93                      ang_max[i] - sec_ang : angs[i]);
94
95          }
96
97          // Publicamos en el topic del robot
98
```

```
99          youbot.PublishMessage();
100
101         // Imprimimos en pantalla los ángulos que le hemos pasado al robot
102
103         std::cout << "Published angles: " << std::endl;
104         for(unsigned int i = 0; i < 5; ++i)
105         {
106             std::cout << "joint #" << (i+1) << ": " << youbot.joint_positions[i] << std
                ::endl;
107         }
108         std::cout << "-------------------------------" << std::endl;
109
110         // Frecuencia del bucle: 20 Hz
111         ros::Duration(0.05).sleep();
112         ros::spinOnce();
113     }
114
115     return 0;
116 }
117
118 double NormalizeAngle(double angle)
119 {
120     while(angle < 0)
121     {
122         angle += 2*dfv::pi;
123     }
124     while(angle > 2*dfv::pi)
125     {
126         angle -= 2*dfv::pi;
127     }
128     return angle;
129 }
130
131 std::vector<double> GetAngles(xsens::SensorSubscriberList& sensors)
132 {
133     std::vector<double> angles(5);
134
135     dfv::Quaternion q0 = sensors.GetOriQuat(0);
136     dfv::Quaternion q1 = sensors.GetOriQuat(1);
137     dfv::Quaternion q2 = sensors.GetOriQuat(2);
138
139     dfv::Quaternion q0p = dfv::Quaternion::GetRotationQuaternion(dfv::Vector3::i.
            GetRotated(q0), dfv::pi / 4.0);
140     dfv::Quaternion q1p = dfv::Quaternion::GetRotationQuaternion(dfv::Vector3::i.
            GetRotated(q1), dfv::pi / 4.0);
141     dfv::Quaternion q2p = dfv::Quaternion::GetRotationQuaternion(dfv::Vector3::i.
            GetRotated(q2), dfv::pi / 4.0);
142
143     dfv::Quaternion q01 = dfv::Quaternion::GetDifference(q0p*q0, q1p*q1);
144     dfv::Quaternion q12 = dfv::Quaternion::GetDifference(q1p*q1, q2p*q2);
145
146     // Cálculo de los ángulos entre cada sensor
147
148     double roll_0;
149     double pitch_0;
150     double yaw_0;
151     q0.GetRPY(roll_0, pitch_0, yaw_0, 1);
152     if(fabs(roll_0) > dfv::pi/2.0)
```

```
153     {
154         q0.GetRPY(roll_0, pitch_0, yaw_0, 2);
155     }
156 
157     double roll_01;
158     double pitch_01;
159     double yaw_01;
160     q01.GetRPY(roll_01, pitch_01, yaw_01, 1);
161     if(fabs(roll_01) > dfv::pi/2.0)
162     {
163         q01.GetRPY(roll_01, pitch_01, yaw_01, 2);
164     }
165 
166     double roll_12;
167     double pitch_12;
168     double yaw_12;
169     q12.GetRPY(roll_12, pitch_12, yaw_12, 1);
170     if(fabs(roll_12) > dfv::pi/2.0)
171     {
172         q12.GetRPY(roll_12, pitch_12, yaw_12, 2);
173     }
174 
175     // Adaptación de los ángulos obtenidos
176     // teniendo en cuenta los offsets de cada
177     // articulación
178 
179     double offsets[] = {2.9, -0.4, -2.5, 1.5, 1.5};
180 
181     double angs[5];
182     angles[0] = offsets[0] + NormalizeAngle(-yaw_0);
183     angles[1] = offsets[1] + 1.0 * (-pitch_0);
184     angles[2] = offsets[2] + 1.0 * (yaw_01);
185     angles[3] = offsets[3] + 1.0 * (yaw_12);
186     angles[4] = offsets[4] ;//+ 2.0 * (-roll_01);
187 
188     return angles;
189 }
```

3.2. youbot.h

```
/*
 * Clase Youbot. Encapsula el mecanismo
 * de publicación de los mensajes necesarios
 * para mover las articulaciones del
 * robot Youbot.
 *
 * Autor: Daniel Fernández Villanueva
 * Mayo de 2013
 *
 */

#ifndef YOUBOT_H
#define YOUBOT_H

#include <string>
#include <cstdlib>
#include <ros/ros.h>
#include <brics_actuator/JointPositions.h>

class Youbot
{
    public:
        Youbot(ros::NodeHandle& node_handle_,
               std::string arm_topic_name_ = "arm_1/arm_controller/position_command",
               std::string gripper_topic_name_ = "arm_1/gripper_controller/
                   position_command");
        ~Youbot();

        double joint_positions[5];
        double gripper_positions[2];
        void PublishMessage(bool publish_gripper = false);

    private:
        ros::NodeHandle& node_handle;
        ros::Publisher arm_publisher;
        ros::Publisher gripper_publisher;

        std::vector<brics_actuator::JointValue> v_joint_values;
        std::vector<brics_actuator::JointValue> v_gripper_values;
};

#endif
```

3.3. youbot.cpp

```
#include <youbot_controller/youbot.h>

Youbot::Youbot(ros::NodeHandle& node_handle_,
               std::string arm_topic_name_,
               std::string gripper_topic_name):
    node_handle(node_handle_)
{
    this->arm_publisher =
        this->node_handle.advertise<brics_actuator::JointPositions>(arm_topic_name_, 1)
            ;
    this->gripper_publisher =
        this->node_handle.advertise<brics_actuator::JointPositions>(gripper_topic_name,
             1);

        this->v_joint_values.resize(5);
        for(int i = 0; i < 5; ++i)
        {
            std::stringstream ss;
            ss << "arm_joint_" << (i+1);
            v_joint_values[i].joint_uri = ss.str();
            v_joint_values[i].unit = std::string("rad");
            v_joint_values[i].value = 0.0;
        }

        this->v_gripper_values.resize(2);
        v_gripper_values[0].joint_uri = "gripper_finger_joint_l";
        v_gripper_values[0].unit = std::string("m");
        v_gripper_values[0].value = 0.001;

        v_gripper_values[1].joint_uri = "gripper_finger_joint_r";
        v_gripper_values[1].unit = std::string("m");
        v_gripper_values[1].value = 0.001;

        this->gripper_positions[0] = 0.01;
        this->gripper_positions[1] = 0.01;

}

Youbot::~Youbot()
{

}

void Youbot::PublishMessage(bool publish_gripper)
{
    brics_actuator::JointPositions msg;
    for(int i = 0; i < 5; ++i)
    {
        v_joint_values[i].value = this->joint_positions[i];
    }
    msg.positions = v_joint_values;
    this->arm_publisher.publish(msg);

    // Publicar las posiciones del gripper resulta en una
    // bajada importante en el rendimiento del robot.
```

```
54      // Se recomienda no pubicar las posiciones del gripper
55      // al mismo ritmo que las del brazo
56      if(publish_gripper)
57      {
58          brics_actuator::JointPositions gripper_msg;
59          v_gripper_values[0].value = this->gripper_positions[0];
60          v_gripper_values[1].value = this->gripper_positions[1];
61          gripper_msg.positions = v_gripper_values;
62
63          this->gripper_publisher.publish(gripper_msg);
64      }
65  }
```

Capítulo 4

PAQUETE arm_visualizer

4.1. arm_visualizer.h

```
#include <ros/ros.h>
#include <dfv/dfv.h>
#include <xsens_driver/xsens_sensor_subscriber.h>
#include <arm_visualizer/gazebo_model_list.h>

int main(int argc, char** argv)
{
    ros::init(argc, argv, "arm_visualizer");
    ros::NodeHandle node_handle;

    xsens::SensorSubscriberList sensors(node_handle);

    gazebo::CModelList arm(node_handle);

    double arm_length = 0.30;
    double forearm_length = 0.25;
    double hand_length = 0.17;

    arm.AddModel("arm", dfv::Vector3(0.0,0.0,0.0), "model/arm.xml");
    arm.AddModel("forearm", dfv::Vector3(0.0,0.0,0.0), "model/forearm.xml");
    arm.AddModel("hand", dfv::Vector3(0.0,0.0,0.0), "model/hand.xml");
    arm.Spawn();

    arm.SetJoint(0, 1, dfv::Vector3(arm_length, 0.0, 0.0));
    arm.SetJoint(1, 2, dfv::Vector3(forearm_length, 0.0, 0.0));

    while(ros::ok())
    {
        for(unsigned int i = 0; i < arm.GetCount(); i++)
        {
            arm.SetOrientation(i, sensors.GetOriQuat(i));
        }

        /* Código de prueba con un sensor
        arm.SetOrientation(0, sensors.GetOriQuat(0));
        arm.SetOrientation(1, dfv::Quaternion::identity);
        arm.SetOrientation(2, sensors.GetOriQuat(0).GetConjugate());
        std::cout << "ORI: " << sensors.GetOriQuat(0) << std::endl;
        */
        arm.PublishMessage();
```

```
41          ros::spinOnce();
42          //ros::Duration(0.1).sleep();
43      }
44
45      return 0;
46  }
```

4.2. gazebo_model.h

```
#ifndef GAZEBO_MODEL_H
#define GAZEBO_MODEL_H

#include <ros/ros.h>
#include <dfv/dfv.h>
#include <gazebo_msgs/SetModelState.h>
#include <gazebo_msgs/SpawnModel.h>
#include <urdf/model.h>
#include <fstream>

namespace gazebo
{
    class CModel
    {
        public:
            CModel(ros::NodeHandle& node_handle_,
                   ros::ServiceClient& set_state_client_,
                   ros::ServiceClient& spawn_model_client_);
            ~CModel();

            //bool              SubscribeToTopic(std::string topic_name);
            //void              SetModelState(const test3_xbus::MTQuaternion::ConstPtr&
                msg);
            void              Spawn();
            bool              ReadUrdf(std::string filename);

            void              SetName(std::string name);
            void              SetPosition(dfv::Vector3 position);
            void              SetOrientation(dfv::Quaternion orientation);
            void              PublishMessage();
            void              SetParent(gazebo::CModel* lp_parent, dfv::Vector3
                parent_joint_position);
            dfv::Vector3      GetPosition();
            dfv::Quaternion   GetOrientation();

            void              SetId(int id);
            int               GetId();

            static int        count;

        private:
            std::string       name;
            dfv::Quaternion   orientation;
            dfv::Vector3      position;
            ros::NodeHandle&  node_handle;

            ros::ServiceClient& set_state_client;
            ros::ServiceClient& spawn_model_client;

            std::string       urdf_model;
            CModel*           lp_parent;
            dfv::Vector3      parent_joint_position;

            int               id;

```

```
54      };
55  };
56
57  #endif
```

4.3. gazebo_model.cpp

```
#include "arm_visualizer/gazebo_model.h"

namespace gazebo
{

    int gazebo::CModel::count = 0;

    CModel::CModel(ros::NodeHandle& node_handle_,
                         ros::ServiceClient& set_state_client_,
                         ros::ServiceClient& spawn_model_client_) :
        name("unnamed"),
        node_handle(node_handle_),
        set_state_client(set_state_client_),
        spawn_model_client(spawn_model_client_),
        lp_parent(NULL)
    {
        this->SetId(gazebo::CModel::count);
        gazebo::CModel::count++;
    }

    CModel::~CModel()
    {

    }

    void CModel::Spawn()
    {
        gazebo_msgs::SpawnModel spawn_model_msg;
        spawn_model_msg.request.model_name             = this->name;
        spawn_model_msg.request.model_xml              = this->urdf_model;
        spawn_model_msg.request.initial_pose.position.x = this->position.x;
        spawn_model_msg.request.initial_pose.position.y = this->position.y;
        spawn_model_msg.request.initial_pose.position.z = this->position.z;
        spawn_model_msg.request.initial_pose.orientation.w = 0.0;
        spawn_model_msg.request.initial_pose.orientation.x = 0.0;
        spawn_model_msg.request.initial_pose.orientation.y = 0.0;
        spawn_model_msg.request.initial_pose.orientation.z = 0.0;
        spawn_model_msg.request.reference_frame        = "world";

        this->spawn_model_client.call(spawn_model_msg);

    }

    bool CModel::ReadUrdf(std::string filename)
    {
        std::fstream xml_file(filename.c_str(), std::fstream::in);
        std::string xml_string;
        if (xml_file.is_open())
        {
            while ( xml_file.good() )
            {
                std::string line;
                std::getline( xml_file, line);
                xml_string += (line + "\n");
            }
```

```
56              xml_file.close();
57              this->urdf_model = xml_string;
58              return true;
59          }
60          else
61          {
62              std::cout << "Could not open file [" << filename << "] for parsing." << std
                    ::endl;
63              return false;
64          }
65      }
66
67      void CModel::SetName(std::string name)
68      {
69          this->name = name;
70      }
71
72      void CModel::SetPosition(dfv::Vector3 position)
73      {
74          this->position = position;
75      }
76
77      void CModel::SetOrientation(dfv::Quaternion orientation)
78      {
79          this->orientation = orientation;
80      }
81
82      void CModel::PublishMessage()
83      {
84          gazebo_msgs::SetModelState set_state_msg;
85
86          set_state_msg.request.model_state.model_name      = this->name;
87          set_state_msg.request.model_state.reference_frame = "world";
88          set_state_msg.request.model_state.pose.position.x = this->GetPosition().x;
89          set_state_msg.request.model_state.pose.position.y = this->GetPosition().y;
90          set_state_msg.request.model_state.pose.position.z = this->GetPosition().z;
91          set_state_msg.request.model_state.pose.orientation.w = this->orientation.w;
92          set_state_msg.request.model_state.pose.orientation.x = this->orientation.x;
93          set_state_msg.request.model_state.pose.orientation.y = this->orientation.y;
94          set_state_msg.request.model_state.pose.orientation.z = this->orientation.z;
95          set_state_msg.request.model_state.twist.linear.x  = 0.0;
96          set_state_msg.request.model_state.twist.linear.y  = 0.0;
97          set_state_msg.request.model_state.twist.linear.z  = 0.0;
98          set_state_msg.request.model_state.twist.angular.x = 0.0;
99          set_state_msg.request.model_state.twist.angular.y = 0.0;
100         set_state_msg.request.model_state.twist.angular.z = 0.0;
101
102         if(!this->set_state_client.call(set_state_msg))
103         {
104             ROS_ERROR("Could not set model state");
105         }
106     }
107
108     void CModel::SetParent(CModel* lp_parent, dfv::Vector3 parent_joint_position)
109     {
110         this->lp_parent = lp_parent;
111         this->parent_joint_position = parent_joint_position;
112     }
```

```
113
114     dfv::Vector3 CModel::GetPosition()
115     {
116         if (this->lp_parent != NULL)
117         {
118             return this->lp_parent->GetPosition() + this->parent_joint_position.
                    GetRotated(this->lp_parent->GetOrientation());
119         }
120         else
121         {
122             return this->position;
123         }
124     }
125
126     dfv::Quaternion CModel::GetOrientation()
127     {
128         return this->orientation;
129     }
130
131
132     void CModel::SetId(int id)
133     {
134         this->id = id;
135     }
136
137     int CModel::GetId()
138     {
139         return this->id;
140     }
141
142 }
```

4.4. gazebo_model_list.h

```
#ifndef GAZEBO_MODEL_LIST_H
#define GAZEBO_MODEL_LIST_H

#include <list>
#include <sstream>
#include "gazebo_model.h"

namespace gazebo
{
    class CModelList
    {
        public:
            CModelList(ros::NodeHandle& node_handle_);
            ~CModelList();

            void AddModel(std::string name, dfv::Vector3 position, std::string
                xml_model);
            //bool GenerateFromTopic(std::string topic_name);
            //void GetConfig(const test3_xbus::MTConfig::ConstPtr& msg);
            void Spawn() const;
            //void SubscribeModelToTopic(unsigned int index, std::string topic_name);
            void SetJoint(unsigned int parent_index, unsigned int child_index, dfv::
                Vector3 parent_joint_position);
            void SetOrientation(unsigned int index, dfv::Quaternion orientation);
            //void EnableGazebo(bool enable);
            //void EnableRviz(bool enable);
            void PublishMessage() const;
            dfv::Vector3 GetPosition(unsigned int index) const;
            unsigned int GetCount() const;

        private:
            std::vector<gazebo::CModel*> model_list;
            ros::NodeHandle&             node_handle;
            ros::ServiceClient           set_state_client;
            ros::ServiceClient           spawn_model_client;

    };
};

#endif
```

4.5. gazebo_model_list.cpp

```
#include "arm_visualizer/gazebo_model_list.h"

namespace gazebo
{

    CModelList::CModelList(ros::NodeHandle& node_handle_):
        node_handle(node_handle_)
    {
        this->set_state_client = node_handle.serviceClient<gazebo_msgs::SetModelState>
            ("/gazebo/set_model_state");
        this->spawn_model_client = node_handle.serviceClient<gazebo_msgs::SpawnModel> (
            "/gazebo/spawn_urdf_model");
    }

    CModelList::~CModelList()
    {
    }

    void CModelList::AddModel(std::string name, dfv::Vector3 position, std::string
        xml_model)
    {
        gazebo::CModel* lp_model = new gazebo::CModel(this->node_handle,
                                                      this->set_state_client,
                                                      this->spawn_model_client);
        lp_model->SetName(name);
        lp_model->SetPosition(position);
        lp_model->ReadUrdf(xml_model);
        this->model_list.push_back(lp_model);
    }

    void CModelList::Spawn() const
    {
        for(unsigned int i = 0; i < this->model_list.size(); i++)
        {
            this->model_list[i]->Spawn();
        }
    }

    void CModelList::SetJoint(unsigned int parent_index,
                              unsigned int child_index,
                              dfv::Vector3 parent_joint_position)
    {
        this->model_list[child_index]->SetParent(this->model_list[parent_index],
            parent_joint_position);
    }

    void CModelList::SetOrientation(unsigned int index, dfv::Quaternion orientation)
    {
        this->model_list[index]->SetOrientation(orientation);
    }

    void CModelList::PublishMessage() const
    {
        for(unsigned int i = 0; i < this->model_list.size(); i++)
        {
```

```
52              this->model_list[i]->PublishMessage();
53          }
54      }
55
56      dfv::Vector3 CModelList::GetPosition(unsigned int index) const
57      {
58          return this->model_list[index]->GetPosition();
59      }
60
61      unsigned int CModelList::GetCount() const
62      {
63          return this->model_list.size();
64      }
65
66  }
```