ANEXO I

Correspondiente al código del análisis de las imágenes de resonancia magnética (IRM)

```
///*************************************************************///
///*                                                           *///
///*                    PROGRAMA PRINCIPAL TFG                 *///
///*                   AUTOR: ANDER LOIDI YARZA                *///
///*                                                           *///
///*        Integración de un sistema de visión 3D en un       *///
///*            entorno robotizado de cirugía                  *///
///*                                                           *///
///*************************************************************///

#include <iostream>
#include <fstream>
#include <math.h>

///Include opencv
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"

///Include PCL
#include <pcl/io/pcd_io.h>
#include <pcl/io/io.h>
#include <pcl/console/parse.h>
#include <pcl/point_cloud.h>
#include <pcl/point_types.h>
#include <pcl/PCLPointCloud2.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/visualization/pcl_visualizer.h>

using namespace std;
using namespace cv;
using namespace pcl;


///VARIABLES GLOBALES
int seleccionPlano = 0;
int seleccionVista = 0;
int seleccionSeccion = 0;
int seleccionContComp = 0;
int tamAxial = 0;
int tamSagital = 0;
int tamCoronal = 0;
int navegacion = 0;

vector<Mat> vectorJPG;
vector<Mat> vectorJPGRot;
vector<Mat> vectorAxial;
vector<Mat> vectorSagital;
vector<Mat> vectorCoronal;
vector<Mat> vectorAxialContornos;
vector<Mat> vectorSagitalContornos;
vector<Mat> vectorCoronalContornos;
vector<Mat> vectorAxialCompletas;
vector<Mat> vectorSagitalCompletas;
vector<Mat> vectorCoronalCompletas;
vector<Mat> vectorFosasNasalesI;
vector<Mat> vectorFosasNasalesD;
vector<Mat> vectorContornoNariz;
vector<Mat> vectorCerebro;
vector<Mat> vectorContornoCerebro;

pcl::PointCloud<pcl::PointXYZ> nube, nubeF, nubeFosaI, nubeFosaD;
pcl::visualization::PCLVisualizer viewer("SISTEMA NAVEGACION TRIDIMENSIONAL");


///*************************************************************///
///*                                                           *///
///* Autor: Ander Loidi Yarza                                  *///
///* Nombre: imprimeAyuda                                      *///
///* Función: Saca por pantalla información relevante al modo  *///
```

```
71  ///*          de uso para una correcta inicialización del eje- *///
72  ///*          cutable.                                          *///
73  ///* Recibe: El nombre del programa/ejecutable.                *///
74  ///*                                                           *///
75  ///*************************************************************///
76  void imprimeAyuda(char *nombrePrograma)
77  {
78      cout << endl;
79      cout << "Error de introducion de datos" << endl;
80      cout << "Modo de uso: " << nombrePrograma << " numero_imagenes" << endl;
81      cout << endl;
82      exit(-1);
83  }
84
85  ///*************************************************************///
86  ///*                                                           *///
87  ///* Autor: Ander Loidi Yarza                                  *///
88  ///* Nombre: cargaImagenes                                     *///
89  ///* Función: Da la opción de cargar nuevas imágenes de reso-  *///
90  ///*          nancia magnética en formato JPG que almacenará   *///
91  ///*          en un vector.                                    *///
92  ///* Recibe: El número de imágenes de las que está formada la  *///
93  ///*         serie de imágenes.                                *///
94  ///* Devuelve: El valor de 1,2 o 3 en función de la selección. *///
95  ///*                                                           *///
96  ///*************************************************************///
97  int cargaImagenes(int numeroImagenes)
98  {
99      int seleccionCarga = 0;
100     system("clear");
101     cout << endl;
102     cout << "*************************    CARGA    *************************" << endl;
103     cout << "*  Desea cargar las imagenes:                                  *" << endl;
104     cout << "*      1 -> Si                                                 *" << endl;
105     cout << "*      2 -> No                                                 *" << endl;
106     cout << "*      3 -> Salir                                              *" << endl;
107     cout << "************************************************************************" << endl;
108     cout << endl;
109     cout << "SELECCION CARGA: ";
110     cin >> seleccionCarga;
111     cout << endl;
112
113     if(seleccionCarga == 1)
114     {
115         vectorJPG.clear();
116
117         for(int i = 0; i<numeroImagenes; i++)
118         {
119             string numero = to_string(i+1);
120             string jpg = "../imagenes/JPG/" + numero + ".jpg";
121
122             Mat imagen = imread(jpg);
123             if (!imagen.data)
124             {
125                 std::cout << "Archivo de imagen " << numero << " no encontrado" << endl;
126                 exit(-1);
127             }
128             else
129             {
130                 vectorJPG.push_back(imagen);
131             }
132         }
133
134         cout << "Imagenes JPG cargadas" << endl;
135     }
136     else if(seleccionCarga == 2)
137     {
138         cout << "Imagenes JPG existentes" << endl;
139     }
```

```
140     else if(seleccionCarga == 3)
141     {
142         exit(-1);
143     }
144 
145     return seleccionCarga;
146 }
147 
148 ///*************************************************************///
149 ///*                                                           *///
150 ///* Autor: Ander Loidi Yarza                                  *///
151 ///* Nombre: rotar                                             *///
152 ///* Función: Rota una imagen con un ángulo determinado.       *///
153 ///* Recibe: La imágenes y el ángulo de giro en grados.        *///
154 ///* Devuelve: La imagen rotada.                               *///
155 ///*                                                           *///
156 ///*************************************************************///
157 Mat rotar(Mat imagen, double angulo)
158 {
159     Mat rotada;
160     Point2f punto(imagen.cols/2., imagen.rows/2.);
161     Mat rotacion = getRotationMatrix2D(punto, angulo, 1.0);
162     warpAffine(imagen, rotada, rotacion, Size(imagen.cols, imagen.rows));
163     return rotada;
164 }
165 
166 ///*************************************************************///
167 ///*                                                           *///
168 ///* Autor: Ander Loidi Yarza                                  *///
169 ///* Nombre: generaPlanosAnatomicos                            *///
170 ///* Función: Crea las series de imágenes de los planos sagi-  *///
171 ///*          tal y coronal a partir del axial (Que es la que  *///
172 ///*          proporciona el Hospital Clínico) y los almacena  *///
173 ///*          de manera global.                                *///
174 ///*                                                           *///
175 ///*************************************************************///
176 void generaPlanosAnatomicos()
177 {
178     vectorAxial.clear();
179     vectorSagital.clear();
180     vectorCoronal.clear();
181 
182     Mat imagen = Mat(vectorJPG[1].rows, vectorJPG[1].cols, CV_8UC1);
183     Mat imagenAxial = Mat(vectorJPG[1].rows, vectorJPG[1].cols, CV_8UC1);
184     Mat imagenSagital = Mat(vectorJPG[1].rows, vectorJPG[1].cols, CV_8UC1);
185     Mat imagenCoronal = Mat(vectorJPG[1].rows, vectorJPG[1].cols, CV_8UC1);
186 
187     int fotosTotales = vectorJPG.size();
188     int filasTotales = vectorJPG[1].rows;
189     int columnasTotales = vectorJPG[1].cols;
190 
191     //AXIAL
192     for(int f = 0; f < fotosTotales; ++f)
193     {
194         for (int fil = 0; fil < filasTotales; ++fil)
195         {
196             for (int col = 1; col < columnasTotales; ++col)
197             {
198                 Vec3b pixel = vectorJPG[f].at<Vec3b>(fil, col);
199                 uchar B = pixel[0];
200                 uchar G = pixel[1];
201                 uchar R = pixel[2];
202                 imagenAxial.at<uchar>(fil, col) = (B + G + R) / 3;
203             }
204         }
205 
206         Mat rot;
207         rot = rotar(imagenAxial, 180);
208         vectorAxial.push_back(rot);
209 
```

```
210             string numero = to_string(f+1);
211             string jpg = "../imagenes/axial/axial"+ numero + ".jpg";
212             //         resize(vectorAxial[f], vectorAxial[f], Size(vectorAxial[f].rows*2,
vectorAxial[f].cols*2));
213             imwrite(jpg, vectorAxial[f]);
214             //         imshow("Transversal", imagenesTransversal[f]);
215             //         waitKey(3);
216         }
217         cout << "Plano axial creado" << endl;
218 
219         //SAGITAL
220         for(int col = 0; col<columnasTotales; ++col)
221         {
222             for(int f = 0; f<fotosTotales; ++f)
223             {
224                 for(int fil = 0; fil<filasTotales; ++fil)
225                 {
226                     Vec3b pixel = vectorJPG[f].at<Vec3b>(fil, col);
227                     uchar B = pixel[0];
228                     uchar G = pixel[1];
229                     uchar R = pixel[2];
230                     imagenSagital.at<uchar>(fil, f) = (B + G + R) / 3;
231                 }
232             }
233 
234             Mat rot;
235             rot = rotar(imagenSagital, 90);
236             vectorSagital.push_back(rot);
237 
238             string numero = to_string(col+1);
239             string jpg = "../imagenes/sagital/sagital"+ numero + ".jpg";
240             //         resize(vectorSagital[col], vectorSagital[col],
Size(vectorSagital[col].rows*2, vectorSagital[col].cols*2));
241             imwrite(jpg, vectorSagital[col]);
242             //         imshow("Sagital", imagenesSagital[col]);
243             //         waitKey(3);
244         }
245         cout << "Plano sagital creado" << endl;
246 
247         //CORONAL
248         Mat JPGRotada;
249         for(int i=0; i<fotosTotales; ++i)
250         {
251             JPGRotada = rotar(vectorJPG[i], 90);
252             vectorJPGRot.push_back(JPGRotada);
253         }
254         for(int col = 0; col<columnasTotales; ++col)
255         {
256             for(int f = 0; f<fotosTotales; ++f)
257             {
258                 for(int fil = 0; fil<filasTotales; ++fil)
259                 {
260                     Vec3b pixel = vectorJPGRot[f].at<Vec3b>(fil, col);
261 
262                     uchar B = pixel[0];
263                     uchar G = pixel[1];
264                     uchar R = pixel[2];
265                     imagenCoronal.at<uchar>(fil, f) = (B + G + R) / 3;
266                 }
267             }
268             Mat rot;
269             rot = rotar(imagenCoronal, 90);
270             vectorCoronal.push_back(rot);
271 
272             string numero = to_string(col+1);
273             string jpg = "../imagenes/coronal/coronal"+ numero + ".jpg";
274 //           resize(vectorCoronal[col], vectorCoronal[col],
Size(vectorCoronal[col].rows*2, vectorCoronal[col].cols*2));
275             imwrite(jpg, vectorCoronal[col]);
276 
```

```
277 //        imshow("Coronal", imagenesCoronal[col]);
278 //        waitKey(3);
279     }
280     cout << "Plano coronal creado" << endl;
281 }
282
283 ///*************************************************************///
284 ///*                                                           *///
285 ///* Autor: Ander Loidi Yarza                                  *///
286 ///* Nombre: creaImagenNubes                                   *///
287 ///* Función: Almacena en un vector el conjunto de imágenes ne-*///
288 ///*          cesarias para la posterior generación de la nube *///
289 ///*          de puntos.                                       *///
290 ///* Recibe: El vector que contiene las imágenes de un plano y *///
291 ///*         el tipo de imágenes que debe generar. 1 para con- *///
292 ///*         tornos y 2 para imagen con información completa.  *///
293 ///* Devuelve: El vector con las imágenes generadas.           *///
294 ///*                                                           *///
295 ///*************************************************************///
296 vector<Mat> creaImagenNubes(vector<Mat> vectorPlano, int tipo)
297 {
298     vector<Mat> vectorImagenes;
299     vectorImagenes.clear();
300
301     for(int i=0; i<vectorPlano.size(); ++i)
302     {
303         Mat imagenTh;
304         Mat salidaContornos;
305         Mat salida;
306
307 //        namedWindow("Origianl", WINDOW_AUTOSIZE);
308 //        namedWindow("Binarizada", WINDOW_AUTOSIZE);
309 //        namedWindow("Floodfill", WINDOW_AUTOSIZE);
310 //        namedWindow("Inversa", WINDOW_AUTOSIZE);
311 //        namedWindow("Combinada", WINDOW_AUTOSIZE);
312 //        namedWindow("BinarizadaC", WINDOW_AUTOSIZE);
313 //        namedWindow("Contornos", WINDOW_AUTOSIZE);
314 //        namedWindow("ContornosC", WINDOW_AUTOSIZE);
315 //
316 //        moveWindow("Original", 0, 400);
317 //        moveWindow("Binarizada", 450, 400);
318 //        moveWindow("Floodfill", 800, 400);
319 //        moveWindow("Inversa", 1250, 500);
320 //        moveWindow("Cobinada", 1250, 100);
321 //        moveWindow("BinarizadaC", 1600, 100);
322 //        moveWindow("Contornos", 1250, 0);
323 //        moveWindow("Contornos", 1600, 0);
324
325 //        imshow("Original",vectorPlano[i]);
326
327         if(tipo == 1)
328         {
329             threshold(vectorPlano[i], imagenTh, 15, 200, THRESH_BINARY_INV);
330 //            imshow("Binarizada", imagenTh);
331
332             //Rellenamos la imagen
333             Mat imagenRellena = imagenTh.clone();
334             floodFill(imagenRellena, Point(0,0), Scalar(255));
335 //            imshow("Floodfill",imagenRellena);
336
337             //Invertimos la imagen rellena
338             Mat ImagenRellenaInv;
339             bitwise_not(imagenRellena, ImagenRellenaInv);
340 //            imshow("Inversa", ImagenRellenaInv);
341
342             //Combinamos ambas imagenes
343             Mat imagenCombinada;
344             imagenCombinada = (imagenTh | ImagenRellenaInv);
345             threshold(imagenCombinada, imagenCombinada, 245, 255, CV_THRESH_BINARY);
346 //            imshow("Combinada", imagenCombinada);
```

```
347             salidaContornos = imagenCombinada.clone();
348 
349         }
350         else if(tipo == 2)
351         {
352             threshold(vectorPlano[i], imagenTh, 30, 180, THRESH_BINARY_INV);
353 //            imshow("BinarizadaC", imagenTh);
354             salidaContornos = imagenTh.clone();
355         }
356 
357 
358         //Buscamos los contornos
359         vector<vector<Point> > contornos;
360         findContours( salidaContornos, contornos, CV_RETR_LIST, CV_CHAIN_APPROX_NONE );
361 
362         Mat imagenContornos(vectorPlano[i].size(), CV_8UC3, Scalar(0,0,0));
363         Scalar colors[3];
364         colors[0] = Scalar(255, 0, 0);
365         colors[1] = Scalar(0, 255, 0);
366         colors[2] = Scalar(0, 0, 255);
367         for (size_t idx = 0; idx < contornos.size(); idx++)
368         {
369             cv::drawContours(imagenContornos, contornos, idx, colors[idx % 3]);
370         }
371 //          imshow("Contornos", imagenContornos);
372 //          waitKey(0);
373         salida = imagenContornos.clone();
374         cvtColor(salida, salida, CV_BGR2GRAY);
375         threshold(salida, salida, 0, 255, CV_THRESH_BINARY);
376 
377         if (tipo == 1)
378         {
379             vectorImagenes.push_back(salida);
380         }
381         else if (tipo == 2)
382         {
383             vectorImagenes.push_back(salida);
384         }
385 
386     }
387 
388     return vectorImagenes;
389 
390 }
391 
392 ///*****************************************************************///
393 ///*                                                               *///
394 ///* Autor: Ander Loidi Yarza                                      *///
395 ///* Nombre: crePCD                                                *///
396 ///* Función: Crea la nube de puntos para cada una de las imáge-*///
397 ///*          nes de cada uno de los planos anatómicos.            *///
398 ///* Recibe: El vector que contiene las imágenes de un plano,un *///
399 ///*         indicador del plano que se trata (1-axial, 2-sagi- *///
400 ///*         tal y 3-coronal) y el tipo de nube que debe generar*///
401 ///*         (1-contorno y 2-cpmpleta).                            *///
402 ///*                                                               *///
403 ///*****************************************************************///
404 void creaPCD(vector<Mat> imagen, int plano, int tipo)
405 {
406     string numero;
407     string nombreFichero;
408 
409     int numArchivo = 0;
410 
411     for(int i = 0; i<imagen.size(); ++i)
412     {
413         int puntosImagen = 0;
414 
415         for(size_t fil=2; fil<imagen[i].rows-2; ++fil)
416         {
```

```
417                 for(size_t col=2; col<(imagen[i].cols/4)-2; ++col)
418                 {
419                     if(imagen[i].at<float>(fil,col)!=0)
420                     {
421                         puntosImagen = puntosImagen + 1;
422                     }
423                 }
424             }
425 
426             if(puntosImagen != 0 )
427             {
428                 ++numArchivo;
429                 numero = to_string(numArchivo);
430 
431                 if(plano == 1)
432                 {
433                     if(tipo == 1)
434                         nombreFichero = "../imagenes/axial/axialContorno_" + numero + ".pcd";
435                     if(tipo == 2)
436                         nombreFichero= "../imagenes/axial/axialCompleta_" + numero + ".pcd";
437                 }
438                 if(plano == 2)
439                 {
440                     if(tipo == 1)
441                         nombreFichero = "../imagenes/sagital/sagitalContorno_" + numero +
".pcd";
442                     if(tipo == 2)
443                         nombreFichero = "../imagenes/sagital/sagitalCompleta_" + numero +
".pcd";
444                 }
445                 if(plano == 3)
446                 {
447                     if(tipo == 1)
448                         nombreFichero = "../imagenes/coronal/coronalContorno_" + numero +
".pcd";
449                     if(tipo == 2)
450                         nombreFichero = "../imagenes/coronal/coronalCompleta_" + numero +
".pcd";
451                 }
452 
453                 ofstream fichero(nombreFichero);
454 
455                 fichero << "# .PCD v0.7 - Point Cloud Data file format" << endl;
456                 fichero << "VERSION 0.7" << endl;
457                 fichero << "FIELDS x y z" << endl;
458                 fichero << "SIZE 4 4 4" << endl;
459                 fichero << "TYPE F F F" << endl;
460                 fichero << "COUNT 1 1 1" << endl;
461                 fichero << "WIDTH " << puntosImagen << endl;
462                 fichero << "HEIGHT 1" << endl;
463                 fichero << "VIEWPOINT 0 0 0 1 0 0 0" << endl;
464                 fichero << "POINTS " << puntosImagen << endl;
465                 fichero << "DATA ascii" << endl;
466 
467                 for(int fil=2; fil<imagen[i].rows-2; ++fil)
468                 {
469                     for(int col=2; col<(imagen[i].cols/4)-2; ++col)
470                     {
471                         if(imagen[i].at<float>(fil,col)!=0)
472                         {
473                             fichero <<  fil*0.75  << " " << col*4*0.75 << " " << i*0.8 << endl;
                                //0.75 0.75 0.8 PARA QUE TENGA DIMENSIONES REALES
474                         }
475                     }
476                 }
477 
478                 fichero.close();
479             }
480         }
481 
```

```
482     ofstream fichero;
483
484     if(plano == 1)
485     {
486         tamAxial = numArchivo;
487
488         if(tipo == 1)
489         {
490             fichero.open("axialContorno.txt", ofstream::trunc);
491             fichero << tamAxial;
492         }
493         if(tipo == 2)
494         {
495             fichero.open("axialCompleta.txt", ofstream::trunc);
496             fichero << tamAxial;
497         }
498     }
499     if(plano == 2)
500     {
501         tamSagital = numArchivo;
502
503         if(tipo == 1)
504         {
505             fichero.open("sagitalContorno.txt", ofstream::trunc);
506             fichero << tamSagital;
507         }
508         if(tipo == 2)
509         {
510             fichero.open("sagitalCompleta.txt", ofstream::trunc);
511             fichero << tamSagital;
512         }
513     }
514
515     if(plano == 3)
516     {
517         tamCoronal = numArchivo;
518
519         if(tipo == 1)
520         {
521             fichero.open("coronalContorno.txt", ofstream::trunc);
522             fichero << tamCoronal;
523         }
524         if(tipo == 2)
525         {
526             fichero.open("coronalCompleta.txt", ofstream::trunc);
527             fichero << tamCoronal;
528         }
529     }
530
531     fichero.close();
532 }
533
534 ///*************************************************************///
535 ///*                                                           *///
536 ///* Autor: Ander Loidi Yarza                                  *///
537 ///* Nombre: vaciado                                           *///
538 ///* Función: Vacía el posible contenido de vectores críticos. *///
539 ///*                                                           *///
540 ///*************************************************************///
541 void vaciado()
542 {
543     vectorJPG.clear();
544     vectorJPGRot.clear();
545     vectorAxial.clear();
546     vectorSagital.clear();
547     vectorCoronal.clear();
548 }
549
550 ///*************************************************************///
551 ///*                                                           *///
```

```
552  ///* Autor: Ander Loidi Yarza                                    *///
553  ///* Nombre: menuSeleccionPlano                                  *///
554  ///* Función: Muestra una serie de menos por pantalla para se- *///
555  ///*          leccionar la vista tridimensional deseada.         *///
556  ///* Devuelve: Valor 0 para finalizar.                           *///
557  ///*                                                             *///
558  ///*************************************************************///
559
560  int menuSeleccionPlano()
561  {
562  menuPlano:
563      seleccionPlano = 0;
564      while(seleccionPlano < 1 || seleccionPlano > 4)
565      {
566          system("clear");
567          cout << endl;
568          cout << "*************************     PLANO     **************************" <<
endl;
569          cout << "*  Seleccione el plano anatomico que quiere visualizar:         *" << endl;
570          cout << "*      1 -> Plano axial                                          *" <<
endl;
571          cout << "*      2 -> Plano sagital                                        *" <<
endl;
572          cout << "*      3 -> Plano coronal                                        *" <<
endl;
573          cout << "*      4 -> Salir                                                *" <<
endl;
574          cout << "*********************************************************************" <<
endl;
575          cout << endl;
576          cout << "SELECCION PLANO: ";
577          cin >> seleccionPlano;
578          cout << endl;
579
580          if(seleccionPlano == 4)
581              return 0;
582
583          if(seleccionPlano < 1 || seleccionPlano > 4)
584              cout << "    *****            Â¡Â¡SELECCION ERRONEA!!            *****    "
<< endl;
585      }
586
587  menuVista:
588      seleccionVista = 0;
589      while(seleccionVista < 1 || seleccionVista > 5)
590      {
591          system("clear");
592          cout << endl;
593          cout << "*************************     VISTA     **************************" <<
endl;
594          cout << "*  Seleccione la vista de la seccion que quiere visualizar:     *" << endl;
595          cout << "*      1 -> Vista del volumen completa                           *" <<
endl;
596          cout << "*     2 -> Vista del volumen hasta una seccion                   *" << endl;
597          cout << "*      3 -> Vista  de una seccion individualmente                *" <<
endl;
598          cout << "*      4 -> Menu anterior                                        *" <<
endl;
599          cout << "*      5 -> Salir                                                *" <<
endl;
600          cout << "*********************************************************************" <<
endl;
601          cout << endl;
602          cout << "SELECCION VISTA: ";
603          cin >> seleccionVista;
604          cout << endl;
605
606          if(seleccionVista == 5)
607              return 0;
608
```

```
609             if(seleccionVista < 1 || seleccionVista > 5)
610                 cout << "    *****            Â¡Â¡SELECCION ERRONEA!!            *****    "
<< endl;
611         }
612
613         if(seleccionVista == 4)
614             goto menuPlano;
615
616     menuFormato:
617         seleccionContComp = 0;
618         while(seleccionContComp < 1 || seleccionContComp > 4)
619         {
620             system("clear");
621             cout << endl;
622             cout << "************************    FORMATO    ************************" <<
endl;
623             cout << "*  Seleccione el formato en el que quiere visualizarla:        *" << endl;
624             cout << "*     1 -> Contornos                                           *" <<
endl;
625             cout << "*     2 -> Completa                                            *" <<
endl;
626             cout << "*     3 -> Menu anterior                                       *" <<
endl;
627             cout << "*     4 -> Salir                                               *" <<
endl;
628             cout << "*****************************************************************" <<
endl;
629             cout << endl;
630             cout << "SELECCION FORMATO: ";
631             cin >> seleccionContComp;
632             cout << endl;
633
634             if(seleccionContComp == 4)
635                 return 0;
636
637             if(seleccionContComp < 1 || seleccionContComp > 4)
638                 cout << "    *****            Â¡Â¡SELECCION ERRONEA!!            *****    "
<< endl;
639         }
640
641         if(seleccionContComp == 3)
642             goto menuVista;
643
644         seleccionSeccion = 0;
645         if(seleccionVista == 2 || seleccionVista == 3)
646         {
647             int tam = 0;
648             char cadena[10];
649
650             if(seleccionPlano == 1)
651             {
652                 if(seleccionContComp == 1)
653                 {
654                     ifstream fichero("axialContorno.txt");
655                     fichero >> cadena;
656                     tam = atoi(cadena);
657                 }
658                 if(seleccionContComp == 2)
659                 {
660                     ifstream fichero("axialCompleta.txt");
661                     fichero >> cadena;
662                     tam = atoi(cadena);
663                 }
664             }
665             if(seleccionPlano == 2)
666             {
667                 if(seleccionContComp == 1)
668                 {
669                     ifstream fichero("sagitalContorno.txt");
670                     fichero >> cadena;
```

```
671                 tam = atoi(cadena);
672             }
673             if(seleccionContComp == 2)
674             {
675                 ifstream fichero("sagitalCompleta.txt");
676                 fichero >> cadena;
677                 tam = atoi(cadena);
678             }
679         }
680         if(seleccionPlano == 3)
681         {
682             if(seleccionContComp == 1)
683             {
684                 ifstream fichero("coronalContorno.txt");
685                 fichero >> cadena;
686                 tam = atoi(cadena);
687             }
688             if(seleccionContComp == 2)
689             {
690                 ifstream fichero("coronalCompleta.txt");
691                 fichero >> cadena;
692                 tam = atoi(cadena);
693             }
694         }
695         while(seleccionSeccion == 0 || seleccionSeccion > tam)
696         {
697             system("clear");
698             cout << endl;
699             cout << "************************    SECCION    ************************"
<< endl;
700             cout << "*  Seccion -> Entre 1 y " << tam << "
*" << endl;
701             cout << "*  -1 -> Menu anterior                                        *" <<
endl;
702             cout << "*  -2 -> Salir                                                *"
<< endl;
703             cout << "*************************************************************"
<< endl;
704             cout << endl;
705             cout << "SELECCION SECCION: ";
706             cin >> seleccionSeccion;
707             cout << endl;
708
709             if(seleccionSeccion == 0 || seleccionSeccion > tam)
710                 cout << "    *****          Â¡Â¡SELECCION ERRONEA!!          *****
" << endl;
711             if(seleccionSeccion == -2)
712                 return 0;
713         }
714         if(seleccionSeccion == -1)
715             goto menuFormato;
716     }
717
718     return 1;
719 }
720
721 ///*************************************************************///
722 ///*                                                           *///
723 ///* Autor: Ander Loidi Yarza                                  *///
724 ///* Nombre: generaSeleccion                                   *///
725 ///* Función: Genera la nube de puntos correspondiente a la se-*///
726 ///*          lección del usuario a través de los menús y la   *///
727 ///*          guarda con el nombre "nube.pcd".                 *///
728 ///* Recibe: El plano (1-axial, 2-sagital y 3-coronal) la vis- *///
729 ///*         ta(1-completa, 2-hasta una sección y 3-sección in-*///
730 ///*         dividual), el formato(1-contorno y 2 completa) y  *///
731 ///*         la sección deseada por el usuario.                *///
732 ///*                                                           *///
733 ///*************************************************************///
734 void generaSeleccion(int plano, int vista, int formato, int seccion)
```

```
735  {
736      int tam = 0;
737      char cadena[10];
738      string numero;
739      string nombreFichero;
740
741      if(plano == 1)
742      {
743          if(formato == 1)
744          {
745              ifstream fichero("axialContorno.txt");
746              fichero >> cadena;
747              tam = atoi(cadena);
748          }
749          if(formato == 2)
750          {
751              ifstream fichero("axialCompleta.txt");
752              fichero >> cadena;
753              tam = atoi(cadena);
754          }
755      }
756      if(plano == 2)
757      {
758          if(formato == 1)
759          {
760              ifstream fichero("sagitalContorno.txt");
761              fichero >> cadena;
762              tam = atoi(cadena);
763          }
764          if(formato == 2)
765          {
766              ifstream fichero("sagitalCompleta.txt");
767              fichero >> cadena;
768              tam = atoi(cadena);
769          }
770      }
771      if(plano == 3)
772      {
773          if(formato == 1)
774          {
775              ifstream fichero("coronalContorno.txt");
776              fichero >> cadena;
777              tam = atoi(cadena);
778          }
779          if(formato == 2)
780          {
781              ifstream fichero("coronalCompleta.txt");
782              fichero >> cadena;
783              tam = atoi(cadena);
784          }
785      }
786
787      for(int f = 0; f < tam; ++f)
788      {
789          numero = to_string(f+1);
790
791          if(plano == 1 && formato == 1)
792              nombreFichero = "../imagenes/axial/axialContorno_" + numero + ".pcd";
793          if(plano == 1 && formato == 2)
794              nombreFichero = "../imagenes/axial/axialCompleta_" + numero + ".pcd";
795          if(plano == 2 && formato == 1)
796              nombreFichero = "../imagenes/sagital/sagitalContorno_" + numero + ".pcd";
797          if(plano == 2 && formato == 2)
798              nombreFichero = "../imagenes/sagital/sagitalCompleta_" + numero + ".pcd";
799          if(plano == 3 && formato == 1)
800              nombreFichero = "../imagenes/coronal/coronalContorno_" + numero + ".pcd";
801          if(plano == 3 && formato == 2)
802              nombreFichero = "../imagenes/coronal/coronalCompleta_" + numero + ".pcd";
803
804          if (io::loadPCDFile<PointXYZ> (nombreFichero, nube) == -1)
```

```
805          {
806              PCL_ERROR ("No se puede leer el archivo\n");
807              exit(-1);
808          }
809
810          if(vista == 1)
811          {
812  //             cout << "AÃ±adiendo archivo" << nombreFichero << endl;
813              if(f == 0)
814                  nubeF = nube;
815              else
816                  nubeF += nube;
817
818              if(f == tam-1)
819                  io::savePCDFileASCII ("nube.pcd", nubeF);
820
821          }
822          if(vista == 2)
823          {
824              if(f < seccion)
825              {
826  //                cout << "AÃ±adiendo archivo " << nombreFichero  << endl;
827                  if(f == 0)
828                      nubeF = nube;
829                  else
830                      nubeF += nube;
831              }
832
833              if(f == seccion-1)
834                  io::savePCDFileASCII ("nube.pcd", nubeF);
835
836          }
837          if(vista == 3)
838          {
839              if((f+1) == seccion)
840              {
841  //                cout << "AÃ±adiendo archivo" << nombreFichero << endl;
842                  nubeF = nube;
843
844                  io::savePCDFileASCII ("nube.pcd", nubeF);
845              }
846
847          }
848      }
849  }
850
851  ///*************************************************************///
852  ///*                                                           *///
853  ///* Autor: Ander Loidi Yarza                                  *///
854  ///* Nombre: visualizacion                                     *///
855  ///* Función: Visualiza la "nube.pcd" previamente creada, dando*///
856  ///*          opción a incrementar o decrementar el número de  *///
857  ///*          sección a visualizar, cambiar la posición de vi- *///
858  ///*          sualización o salir del menú en tiempo real.     *///
859  ///*                                                           *///
860  ///*************************************************************///
861  void visualizacion()
862  {
863      pcl::PointCloud<pcl::PointXYZ>::Ptr nubeVisualizacion (new
pcl::PointCloud<pcl::PointXYZ>());
864      pcl::PointCloud<pcl::PointXYZ>::Ptr aux (new pcl::PointCloud<pcl::PointXYZ>());
865      int seleccion = 0;
866      io::loadPCDFile("nube.pcd", *nubeVisualizacion);
867
868      if(navegacion == 0)
869      {
870          viewer.resetCameraViewpoint("nube");
871          viewer.addPointCloud(nubeVisualizacion, "nube");
872          viewer.resetCameraViewpoint("nube");
873          viewer.spinOnce(4000);
```

```
874         }
875         else
876         {
877             viewer.resetCameraViewpoint("nube");
878             viewer.updatePointCloud(nubeVisualizacion, "nube");
879             viewer.resetCameraViewpoint("nube");
880             viewer.spinOnce(4000);
881         }
882 
883         while(1)
884         {
885             system("clear");
886             cout << "1 = -  2 = +   3 = mover   4 = salir" << endl;
887             cin >> seleccion;
888             if(seleccion == 1)
889             {
890                 seleccionSeccion = seleccionSeccion - 1;
891             }
892             if(seleccion == 2)
893             {
894                 seleccionSeccion = seleccionSeccion + 1;
895             }
896             if(seleccion == 3)
897             {
898                 viewer.resetCameraViewpoint("nube");
899                 viewer.spinOnce(4000);
900             }
901             if(seleccion != 4)
902             {
903                 generaSeleccion(seleccionPlano, seleccionVista, seleccionContComp,
seleccionSeccion);
904                 io::loadPCDFile("nube.pcd", *nubeVisualizacion);
905                 viewer.updatePointCloud (nubeVisualizacion, "nube");
906                 viewer.spinOnce(1);
907             }
908             if(seleccion == 3)
909             {
910                 break;
911             }
912 
913         }
914         ++navegacion;
915     }
916 
917     ///*************************************************************///
918     ///*                                                           *///
919     ///* Autor: Ander Loidi Yarza                                  *///
920     ///* Nombre: buscoFosasNasales                                 *///
921     ///* Función: Genera la nube de puntos correspondientes a las  *///
922     ///*          fosas nasales:                                   *///
923     ///*             "fosas.pcd" - Ambas fosas nasales juntas.     *///
924     ///*             "fosasI.pcd" - Fosa nasal izquierda.          *///
925     ///*             "fosasD.pcd" - Fosa nasal derecha.            *///
926     ///*             "nariz.pcd" - Nariz.                          *///
927     ///*                                                           *///
928     ///*************************************************************///
929     //BUSCA FOSAS NASALES y NARIZ
930     void buscoFosasNasales()
931     {
932         int tam = 0;
933         char cadena[10];
934         int area = 0;
935         string numero;
936         string nombreFichero;
937         Mat imagenJPG, imagenTh, imagenROI, imagenContorno;
938 
939         cout << "Generando nubes de fosas nasales" << endl;
940 
941         ifstream fichero("axialContorno.txt");
942         fichero >> cadena;
```

```
 943      tam = atoi(cadena);
 944
 945      int puntosFosasNasalesI = 0;
 946      int puntosFosasNasalesD = 0;
 947      int puntosContornoNariz = 0;
 948
 949      for(int f = 0; f < tam; ++f)
 950      {
 951  //        namedWindow("Origianl", WINDOW_AUTOSIZE);
 952  //        namedWindow("GaussianBlur", WINDOW_AUTOSIZE);
 953  //        namedWindow("Binarizada", WINDOW_AUTOSIZE);
 954  //        namedWindow("Cerrada", WINDOW_AUTOSIZE);
 955  //        namedWindow("ROI", WINDOW_AUTOSIZE);
 956  //        namedWindow("ContornosI", WINDOW_AUTOSIZE);
 957  //        namedWindow("ContornosD", WINDOW_AUTOSIZE);
 958  //
 959  //        moveWindow("Original", 0, 400);
 960  //        moveWindow("GaussianBlur", 450, 400);
 961  //        moveWindow("Binarizada", 800, 400);
 962  //        moveWindow("Cerrada", 1250, 500);
 963  //        moveWindow("ROI", 1250, 100);
 964  //        moveWindow("ContornosI", 1600, 100);
 965  //        moveWindow("ContornosD", 2000, 500);
 966
 967
 968          numero = to_string(f+1);
 969
 970          nombreFichero = "../imagenes/axial/axial" + numero + ".jpg";
 971
 972          imagenJPG = imread(nombreFichero, IMREAD_GRAYSCALE);
 973
 974  //        imshow("Original", imagenJPG);
 975
 976          GaussianBlur(imagenJPG, imagenJPG, Size(3,3),5);
 977  //        imshow("GaussianBlur", imagenJPG);
 978
 979          adaptiveThreshold(imagenJPG, imagenTh, 255, ADAPTIVE_THRESH_MEAN_C,
THRESH_BINARY, 105, 1);
 980  //        imshow("Binarizada", imagenTh);
 981          imagenContorno = imagenTh.clone();
 982
 983          //Generamos region de interes
 984          Rect roi(138,245,40,75); //Recta columna, recta fila, anchura, altura
 985          imagenROI = imagenTh(roi);
 986  //        imshow("ROI", imagenROI);
 987          Mat imagenContornoR = imagenROI.clone();
 988          int morph_size = 0;
 989          Mat element;
 990
 991          if(f<40)
 992          {
 993              int morph_size = 0.9;
 994              Mat element = getStructuringElement(MORPH_ELLIPSE, Size(2*morph_size+1,
2*morph_size+1), Point(morph_size, morph_size));
 995              morphologyEx( imagenContornoR, imagenContornoR, MORPH_CLOSE, element);
 996          }
 997  //        imshow("Cerrada", imagenContornoR);
 998
 999          Rect roiC(120,260,76,60);
1000          imagenContorno = imagenContorno(roiC);
1001
1002          morph_size =4;
1003          element = getStructuringElement(MORPH_ELLIPSE, Size(2*morph_size+1,
2*morph_size+1), Point(morph_size, morph_size));
1004          morphologyEx( imagenContorno, imagenContorno, MORPH_CLOSE, element);
1005  //        imshow("Cerrada", imagenContorno);
1006
1007          vector<vector<Point>> contornos;
1008          vector<vector<Point>> contornosValidosI;
1009          vector<vector<Point>> contornosValidosD;
```

```
1010          vector<vector<Point>> contornosExterno;
1011
1012
1013          findContours(imagenContornoR, contornos, CV_RETR_CCOMP, CV_CHAIN_APPROX_NONE);
1014          findContours(imagenContorno, contornosExterno, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_NONE);
1015
1016          Mat imagenContornosI(imagenJPG.size(), CV_8UC3, Scalar(0,0,0));
1017          Mat imagenContornosD(imagenJPG.size(), CV_8UC3, Scalar(0,0,0));
1018          Mat imagenContornosE(imagenJPG.size(), CV_8UC3, Scalar(0,0,0));
1019
1020
1021          vector<Moments> mu(contornos.size());
1022          for( int i = 0; i<contornos.size(); ++i )
1023          {
1024              mu[i] = moments( contornos[i], false );
1025          }
1026
1027          //Obtenemos el centro de cada area
1028          vector<Point2f> mc(contornos.size());
1029          for( int i = 0; i<contornos.size(); ++i)
1030          {
1031              mc[i] = Point2f( mu[i].m10/mu[i].m00, mu[i].m01/mu[i].m00 );
1032          }
1033
1034  //          cout << endl << "   ***   Area imagen " << f << "   ***" << endl;
1035
1036          for(int i = 0; i < contornos.size(); ++i)
1037          {
1038              area = contourArea(contornos[i]);
1039
1040              if(area > 25 && area < 1100 && mc[i].x<20.0)
1041              {
1042                  contornosValidosI.push_back(contornos[i]);
1043                  drawContours(imagenContornosI, contornosValidosI, -1, (0, 0, 255) );
1044  //                cout << "    " << f << " - El area del contorno " << i << " es de " <<
area << " con centro en " << mc[i] << endl;
1045              }
1046              if(area > 30 && area < 1100 && mc[i].x>20.0)
1047              {
1048                  contornosValidosD.push_back(contornos[i]);
1049                  drawContours(imagenContornosD, contornosValidosD, -1, (0, 0, 255) );
1050                  cout << "    " << f << " - El area del contorno " << i << " es de " << area
<< " con centro en " << mc[i] << endl;
1051              }
1052          }
1053
1054          for(int i = 0; i < contornosExterno.size(); ++i)
1055          {
1056              drawContours(imagenContornosE, contornosExterno, -1, (0, 0, 255) );
1057          }
1058  //          imshow("ContornosI", imagenContornosI);
1059  //          imshow("ContornosD", imagenContornosD);
1060  //          waitKey(0);
1061
1062          contornos.clear();
1063          contornosValidosI.clear();
1064          contornosValidosD.clear();
1065          vectorFosasNasalesI.push_back(imagenContornosI);
1066          vectorFosasNasalesD.push_back(imagenContornosD);
1067          vectorContornoNariz.push_back(imagenContornosE);
1068
1069      }
1070
1071      for(int i = 30; i <vectorFosasNasalesI.size()-130; ++i)
1072      {
1073          for(int fil = 2; fil < vectorFosasNasalesI[i].rows-2; ++fil)
1074          {
1075              for(int col = 2; col < (vectorFosasNasalesI[i].cols/2)-2; ++col)
1076              {
```

```
1077                if(vectorFosasNasalesI[i].at<float>(fil, col) != 0)
1078                {
1079                    ++puntosFosasNasalesI;
1080                }
1081            }
1082        }
1083    }
1084    for(int i = 30; i <vectorFosasNasalesD.size()-130; ++i)
1085    {
1086        for(int fil = 2; fil < vectorFosasNasalesD[i].rows-2; ++fil)
1087        {
1088            for(int col = 2; col < (vectorFosasNasalesD[i].cols/2)-2; ++col)
1089            {
1090                if(vectorFosasNasalesD[i].at<float>(fil, col) != 0)
1091                {
1092                    ++puntosFosasNasalesD;
1093                }
1094            }
1095        }
1096    }
1097    for(int i = 30; i <vectorContornoNariz.size()-110; ++i)
1098    {
1099        for(int fil = 2; fil < vectorContornoNariz[i].rows-2; ++fil)
1100        {
1101            for(int col = 2; col < (vectorContornoNariz[i].cols/2)-2; ++col)
1102            {
1103                if(vectorContornoNariz[i].at<float>(fil, col) != 0)
1104                {
1105                    ++puntosContornoNariz;
1106                }
1107            }
1108        }
1109    }
1110
1111    ofstream ficheroFosasNasalesI("fosasI.pcd");
1112    ofstream ficheroFosasNasalesD("fosasD.pcd");
1113    ofstream ficheroContornoNariz("nariz.pcd");
1114
1115    ficheroFosasNasalesI << "# .PCD v0.7 - Point Cloud Data file format" << endl;
1116    ficheroFosasNasalesI << "VERSION 0.7" << endl;
1117    ficheroFosasNasalesI << "FIELDS x y z" << endl;
1118    ficheroFosasNasalesI << "SIZE 4 4 4" << endl;
1119    ficheroFosasNasalesI << "TYPE F F F" << endl;
1120    ficheroFosasNasalesI << "COUNT 1 1 1" << endl;
1121    ficheroFosasNasalesI << "WIDTH " << puntosFosasNasalesI << endl;
1122    ficheroFosasNasalesI << "HEIGHT 1" << endl;
1123    ficheroFosasNasalesI << "VIEWPOINT 0 0 0 1 0 0 0" << endl;
1124    ficheroFosasNasalesI << "POINTS " << puntosFosasNasalesI << endl;
1125    ficheroFosasNasalesI << "DATA ascii" << endl;
1126
1127    ficheroFosasNasalesD << "# .PCD v0.7 - Point Cloud Data file format" << endl;
1128    ficheroFosasNasalesD << "VERSION 0.7" << endl;
1129    ficheroFosasNasalesD << "FIELDS x y z" << endl;
1130    ficheroFosasNasalesD << "SIZE 4 4 4" << endl;
1131    ficheroFosasNasalesD << "TYPE F F F" << endl;
1132    ficheroFosasNasalesD << "COUNT 1 1 1" << endl;
1133    ficheroFosasNasalesD << "WIDTH " << puntosFosasNasalesD << endl;
1134    ficheroFosasNasalesD << "HEIGHT 1" << endl;
1135    ficheroFosasNasalesD << "VIEWPOINT 0 0 0 1 0 0 0" << endl;
1136    ficheroFosasNasalesD << "POINTS " << puntosFosasNasalesD << endl;
1137    ficheroFosasNasalesD << "DATA ascii" << endl;
1138
1139    ficheroContornoNariz << "# .PCD v0.7 - Point Cloud Data file format" << endl;
1140    ficheroContornoNariz << "VERSION 0.7" << endl;
1141    ficheroContornoNariz << "FIELDS x y z" << endl;
1142    ficheroContornoNariz << "SIZE 4 4 4" << endl;
1143    ficheroContornoNariz << "TYPE F F F" << endl;
1144    ficheroContornoNariz << "COUNT 1 1 1" << endl;
1145    ficheroContornoNariz << "WIDTH " << puntosContornoNariz << endl;
1146    ficheroContornoNariz << "HEIGHT 1" << endl;
```

```
1147        ficheroContornoNariz << "VIEWPOINT 0 0 0 1 0 0 0" << endl;
1148        ficheroContornoNariz << "POINTS " << puntosContornoNariz << endl;
1149        ficheroContornoNariz << "DATA ascii" << endl;
1150
1151
1152        for(int i = 30; i <vectorFosasNasalesI.size()-130; ++i)
1153        {
1154            for(int fil = 2; fil < vectorFosasNasalesI[i].rows-2; ++fil)
1155            {
1156                for(int col = 2; col < (vectorFosasNasalesI[i].cols/2)-2; ++col)
1157                {
1158                    if(vectorFosasNasalesI[i].at<float>(fil, col) != 0)
1159                    {
1160                        ficheroFosasNasalesI << (fil*0.75)/2 << " " << col*0.75<< " " <<
1161    i*0.8<< endl;
1162                    }
1163                }
1164            }
1165        }
1166        for(int i = 30; i <vectorFosasNasalesD.size()-130; ++i)
1167        {
1168            for(int fil = 2; fil < vectorFosasNasalesD[i].rows-2; ++fil)
1169            {
1170                for(int col = 2; col < (vectorFosasNasalesD[i].cols/2)-2; ++col)
1171                {
1172                    if(vectorFosasNasalesD[i].at<float>(fil, col) != 0)
1173                    {
1174                        ficheroFosasNasalesD << (fil*0.75)/2 << " " << col*0.75 << " " << i*0.8
1175    << endl;
1176                    }
1177                }
1178            }
1179        }
1180        for(int i = 30; i <vectorContornoNariz.size()-110; ++i)
1181        {
1182            for(int fil = 2; fil < vectorContornoNariz[i].rows-2; ++fil)
1183            {
1184                for(int col = 2; col < (vectorContornoNariz[i].cols/2)-2; ++col)
1185                {
1186                    if(vectorContornoNariz[i].at<float>(fil, col) != 0)
1187                    {
1188                        ficheroContornoNariz << (fil*0.75+15*0.75)/2 << " " <<
1189    (col*0.75-14*0.75) << " " << i*0.8 << endl;
1190                    }
1191                }
1192            }
1193        }
1194
1195        ficheroFosasNasalesI.close();
1196        ficheroFosasNasalesD.close();
1197        ficheroContornoNariz.close();
1198
1199        if (io::loadPCDFile<PointXYZ> ("fosasI.pcd", nubeFosaI) == -1 ||
1200    io::loadPCDFile<PointXYZ> ("fosasD.pcd", nubeFosaD) == -1)
1201        {
1202            PCL_ERROR ("No se puede leer el archivo\n");
1203            exit(-1);
1204        }
1205
1206        nubeF = nubeFosaI;
1207        nubeF += nubeFosaD;
1208
1209        io::savePCDFileASCII ("fosas.pcd", nubeF);
1210    }
1211
1212    ///*************************************************************///
    ///*                                                           *///
    ///* Autor: Ander Loidi Yarza                                  *///
    ///* Nombre: buscoCerebro                                      *///
    ///* Función: Genera la nube de puntos correspondientes a la   *///
```

```
1213  ///*          estructura cerebral:                                  *///
1214  ///*             "cerebro.pcd" - Estructura cerebral.               *///
1215  ///*             "cabeza.pcd" - Cabeza del paciente.                *///
1216  ///*                                                                *///
1217  ///****************************************************************///
1218  void buscoCerebro()
1219  {
1220      int tam = 0;
1221      char cadena[10];
1222      int area = 0;
1223      int puntosCerebro = 0;
1224      int puntosContorno = 0;
1225      string numero;
1226      string nombreFichero;
1227      Mat imagenJPG, imagenTh, cerebroContorno, cerebro;
1228
1229      cout << "Generando nube de cerebro" << endl;
1230
1231      ifstream fichero("sagitalCompleta.txt");
1232      fichero >> cadena;
1233      tam = atoi(cadena);
1234
1235      namedWindow("Origianl", WINDOW_AUTOSIZE);
1236      namedWindow("Binarizada", WINDOW_AUTOSIZE);
1237      namedWindow("Erosionada", WINDOW_AUTOSIZE);
1238      namedWindow("Bis", WINDOW_AUTOSIZE);
1239      namedWindow("Contornos", WINDOW_AUTOSIZE);
1240
1241      moveWindow("Original", 10, 400);
1242      moveWindow("Binarizada", 450, 400);
1243      moveWindow("Erosionada", 800, 400);
1244      moveWindow("Bis", 1250, 400);
1245      moveWindow("Contornos", 2000, 400);
1246
1247      for(int f = 52; f < tam+52; ++f)
1248      {
1249          numero = to_string(f+1);
1250
1251          nombreFichero = "../imagenes/sagital/sagital" + numero + ".jpg";
1252
1253          imagenJPG = imread(nombreFichero, IMREAD_GRAYSCALE);
1254          imshow("Original", imagenJPG);
1255          GaussianBlur(imagenJPG, imagenJPG, Size(1.5,1.5), 4);
1256
1257          threshold(imagenJPG, imagenTh, 50, 254,THRESH_BINARY);
1258
1259  //          adaptiveThreshold(imagenJPG, imagenTh, 255, ADAPTIVE_THRESH_MEAN_C,
THRESH_BINARY, 105, 1);
1260
1261          threshold(imagenJPG, cerebroContorno, 10, 200, THRESH_BINARY_INV);
1262          GaussianBlur(cerebroContorno, cerebroContorno, Size(1.5,1.5), 4);
1263
1264          Mat imagenRellena = cerebroContorno.clone();
1265          floodFill(imagenRellena, Point(0,0), Scalar(255));
1266
1267          //Invertimos la imagen rellena
1268          Mat ImagenRellenaInv;
1269          bitwise_not(imagenRellena, ImagenRellenaInv);
1270
1271          //Combinamos ambas imagenes
1272          Mat imagenCombinada;
1273          imagenCombinada = (cerebroContorno | ImagenRellenaInv);
1274          threshold(imagenCombinada, imagenCombinada, 245, 255, CV_THRESH_BINARY);
1275
1276          cerebroContorno = imagenCombinada.clone();
1277
1278          int morph_size = 5;
1279          Mat element = getStructuringElement(MORPH_ELLIPSE, Size(2*morph_size+1,
2*morph_size+1), Point(morph_size, morph_size));
1280          morphologyEx( cerebroContorno, cerebroContorno, MORPH_OPEN, element);
```

```
1281  //        morphologyEx( cerebroContorno, cerebroContorno, MORPH_CLOSE, element);
1282
1283          morph_size = 0.5;
1284          element = getStructuringElement(MORPH_ELLIPSE, Size(morph_size+1, morph_size+1),
Point(morph_size, morph_size));
1285          morphologyEx( imagenTh, imagenTh, MORPH_CLOSE, element);
1286          imshow("Binarizada", imagenTh);
1287
1288          morph_size = 2;
1289          element = getStructuringElement(MORPH_ELLIPSE, Size(2*morph_size+1,
2*morph_size+1), Point(morph_size, morph_size));
1290          erode(imagenTh, imagenTh, element);
1291          erode(imagenTh, imagenTh, element);
1292          imshow("Erosionada", imagenTh);
1293
1294          element = getStructuringElement(MORPH_ELLIPSE, Size(3.5*morph_size+1,
3.5*morph_size+1), Point(morph_size, morph_size));
1295          morphologyEx( imagenTh, imagenTh, MORPH_CLOSE, element);
1296  //
1297  //        morph_size = 5;
1298          element = getStructuringElement(MORPH_ELLIPSE, Size(3.5*morph_size+1,
3.5*morph_size+1), Point(morph_size, morph_size));
1299          dilate(imagenTh, imagenTh, element);
1300
1301  ////        morphologyEx( imagenTh, imagenTh, MORPH_CLOSE, element);//n
1302          imshow("Bis", imagenTh);
1303
1304          vector<vector<Point>> contornos;
1305          vector<vector<Point>> contornosCabeza;
1306          vector<vector<Point>> contornosValidos;
1307
1308
1309          findContours(imagenTh, contornos, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE);
1310          findContours(cerebroContorno, contornosCabeza, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE);
1311
1312          Mat imagenContornos(imagenJPG.size(), CV_8UC3, Scalar(0,0,0));
1313          Mat imagenContornosC(imagenJPG.size(), CV_8UC3, Scalar(0,0,0));
1314
1315          vector<Moments> mu(contornos.size());
1316          for( int i = 0; i<contornos.size(); ++i )
1317          {
1318              mu[i] = moments( contornos[i], false );
1319          }
1320
1321          //Obtenemos el centro de cada area
1322          vector<Point2f> mc(contornos.size());
1323          for( int i = 0; i<contornos.size(); ++i)
1324          {
1325              mc[i] = Point2f( mu[i].m10/mu[i].m00, mu[i].m01/mu[i].m00 );
1326          }
1327
1328          cout << endl << "    ***    Area imagen " << f << "    ***" << endl;
1329          for(int i = 0; i < contornos.size(); ++i)
1330          {
1331              area = contourArea(contornos[i]);
1332
1333              if((pow((mc[i].x-218.0),2)+pow((mc[i].y-179.0),2))<=50*50)//antes 50**/
1334              {
1335                  contornosValidos.push_back(contornos[i]);
1336                  drawContours(imagenContornos, contornosValidos, -1, Scalar(0,255,0) );
1337                  cout << "    " << f << " - El area del contorno " << i << " es de " << area
<< " con centro en " << mc[i] << endl;
1338              }
1339          }
1340          imshow("Contornos", imagenContornos);
1341          waitKey(0);
1342          for(int i = 0; i < contornosCabeza.size(); ++i)
1343          {
1344              drawContours(imagenContornosC, contornosCabeza, -1, (0, 0, 255) );
```

```
1345  //            cout << "    " << f << " - El area del contorno " << i << " es de " << area
<< " con centro en " << mc[i] << endl;
1346          }
1347
1348          contornos.clear();
1349          contornosValidos.clear();
1350          vectorCerebro.push_back(imagenContornos);
1351          vectorContornoCerebro.push_back(imagenContornosC);
1352
1353      }
1354
1355      for(int i = 0; i <vectorCerebro.size(); ++i)
1356      {
1357          for(int fil = 0; fil < vectorCerebro[i].rows; ++fil)
1358          {
1359              for(int col = 0; col < (vectorCerebro[i].cols)-40; ++col)
1360              {
1361                  if(vectorCerebro[i].at<float>(fil, col) != 0)
1362                  {
1363                      ++puntosCerebro;
1364                  }
1365              }
1366          }
1367      }
1368      for(int i = 0; i <vectorContornoCerebro.size(); ++i)
1369      {
1370          for(int fil = 2; fil < vectorContornoCerebro[i].rows-2; ++fil)
1371          {
1372              for(int col = 1; col < (vectorContornoCerebro[i].cols)-80; ++col)
1373              {
1374                  if(vectorContornoCerebro[i].at<float>(fil, col) != 0)
1375                  {
1376                      ++puntosContorno;
1377                  }
1378              }
1379          }
1380      }
1381
1382      ofstream ficheroCerebro("cerebro.pcd");
1383      ofstream ficheroContorno("cabeza.pcd");
1384
1385      ficheroCerebro << "# .PCD v0.7 - Point Cloud Data file format" << endl;
1386      ficheroCerebro << "VERSION 0.7" << endl;
1387      ficheroCerebro << "FIELDS x y z" << endl;
1388      ficheroCerebro << "SIZE 4 4 4" << endl;
1389      ficheroCerebro << "TYPE F F F" << endl;
1390      ficheroCerebro << "COUNT 1 1 1" << endl;
1391      ficheroCerebro << "WIDTH " << puntosCerebro << endl;
1392      ficheroCerebro << "HEIGHT 1" << endl;
1393      ficheroCerebro << "VIEWPOINT 0 0 0 1 0 0 0" << endl;
1394      ficheroCerebro << "POINTS " << puntosCerebro << endl;
1395      ficheroCerebro << "DATA ascii" << endl;
1396
1397      ficheroContorno << "# .PCD v0.7 - Point Cloud Data file format" << endl;
1398      ficheroContorno << "VERSION 0.7" << endl;
1399      ficheroContorno << "FIELDS x y z" << endl;
1400      ficheroContorno << "SIZE 4 4 4" << endl;
1401      ficheroContorno << "TYPE F F F" << endl;
1402      ficheroContorno << "COUNT 1 1 1" << endl;
1403      ficheroContorno << "WIDTH " << puntosContorno << endl;
1404      ficheroContorno << "HEIGHT 1" << endl;
1405      ficheroContorno << "VIEWPOINT 0 0 0 1 0 0 0" << endl;
1406      ficheroContorno << "POINTS " << puntosContorno << endl;
1407      ficheroContorno << "DATA ascii" << endl;
1408
1409      for(int i = 0; i <vectorCerebro.size(); ++i)
1410      {
1411          for(int fil = 0; fil < vectorCerebro[i].rows; ++fil)
1412          {
1413              for(int col = 0; col < vectorCerebro[i].cols-40; ++col)
```

```
1414                {
1415                    if(vectorCerebro[i].at<float>(fil, col) != 0)
1416                    {
1417                        ficheroCerebro << fil << " " << col << " " << i<< endl;
1418                    }
1419                }
1420            }
1421        }
1422        for(int i = 0; i <vectorContornoCerebro.size(); ++i)
1423        {
1424            for(int fil = 2; fil < vectorContornoCerebro[i].rows-2; ++fil)
1425            {
1426                for(int col = 1; col < vectorContornoCerebro[i].cols-80; ++col)
1427                {
1428                    if(vectorContornoCerebro[i].at<float>(fil, col) != 0)
1429                    {
1430                        ficheroContorno << fil << " " << col << " " << i<< endl;
1431                    }
1432                }
1433            }
1434        }
1435
1436        ficheroCerebro.close();
1437        ficheroContorno.close();
1438
1439    }
1440
1441    ///*************************************************************///
1442    ///*                                                           *///
1443    ///* Autor: Ander Loidi Yarza                                  *///
1444    ///* Nombre: main                                              *///
1445    ///* Función: Encargada de inicializar y llevar a cabo el con- *///
1446    ///*          junto de las funciones.                          *///
1447    ///*                                                           *///
1448    ///*************************************************************///
1449    int main(int argc, char *argv[])
1450    {
1451        int nImagenes = 0;
1452        int carga = 0;
1453        int bucleVisualizacion = 1;
1454
1455        if (console::find_switch (argc, argv, "-h") || console::find_switch (argc, argv,
"--help") || console::find_switch (argc, argv, "--ayuda") || argc != 2)
1456        {
1457            imprimeAyuda(argv[0]);
1458            return 0;
1459        }
1460
1461        nImagenes = atoi(argv[1]);
1462        carga = cargaImagenes(nImagenes);
1463        if(carga == 1)
1464        {
1465            generaPlanosAnatomicos();
1466
1467            vectorAxialContornos = creaImagenNubes(vectorAxial, 1);
1468            vectorAxialCompletas = creaImagenNubes(vectorAxial, 2);
1469            vectorSagitalContornos = creaImagenNubes(vectorSagital, 1);
1470            vectorSagitalCompletas = creaImagenNubes(vectorSagital, 2);
1471            vectorCoronalContornos = creaImagenNubes(vectorCoronal, 1);
1472            vectorCoronalCompletas = creaImagenNubes(vectorCoronal, 2);
1473
1474            creaPCD(vectorAxialContornos, 1, 1);
1475            creaPCD(vectorAxialCompletas, 1, 2);
1476            creaPCD(vectorSagitalContornos, 2, 1);
1477            creaPCD(vectorSagitalCompletas, 2, 2);
1478            creaPCD(vectorCoronalContornos, 3, 1);
1479            creaPCD(vectorCoronalCompletas, 3, 2);
1480
1481        }
1482
```

```
1483        while(bucleVisualizacion == 1)
1484        {
1485
1486            bucleVisualizacion = menuSeleccionPlano();
1487
1488            if(bucleVisualizacion == 1)
1489            {
1490                generaSeleccion(seleccionPlano, seleccionVista, seleccionContComp,
seleccionSeccion);
1491                visualizacion();
1492            }
1493        }
1494
1495        buscoFosasNasales();
1496        buscoCerebro();
1497
1498        cout << endl << "           *****      FIN DE PROGRAMA      *****           " << endl <<
endl;
1499        vaciado();
1500        return 0;
1501    }
```