



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

Teoremas de la alternativa, optimización convexa,
valoración de activos financieros
y
procesamiento de nubes de puntos generadas por
escáner láser.

Autor
Pedro Manuel Flores Crespo

Directores
Manuel Ruiz Galán
Juan Carlos Torres Cantero



Facultad de Ciencias

—
Granada, junio de 2020

Índice general

1. Resumen y palabras clave	5
2. Resumen extendido y palabras clave en inglés	7
3. Introducción	11
4. Objetivos	15
5. Desarrollo	17
I Teoremas de la alternativa, optimización convexa y valoración de activos financieros	19
II Procesamiento de nubes de puntos generadas por escáner láser	21
6. Iterative Closest Point (ICP)	23
6.1. Cuaternios	23
6.1.1. Suma y producto	24
6.1.2. Conjugado, norma e inverso	25
6.1.3. Cuaternios y rotaciones	26
6.2. Cuaternios y optimización	29
6.3. Algoritmo ICP	32
6.4. Ejemplos prácticos	36
7. Cambios en el método: uso de normales	41
7.1. Variación de la normal	41
7.2. Pruebas	43
7.2.1. De puntos clave a todo el modelo	43
7.2.2. De puntos clave a puntos clave	46
7.3. Torre de las gallinas	48
7.4. Sensibilidad y uso en prealineado	52

7.5. Conclusiones	56
8. RANSAC: algoritmo de consenso	57
8.1. Algoritmo	57
8.2. Número de iteraciones	58
8.3. Ejemplos sencillos	60
8.3.1. Pie	61
8.3.2. Torre	61
8.4. Cálculo de los puntos de intersección	62
8.5. Ejemplo despacho	65
8.6. Ejemplo específico	70
9. Banco de pruebas	73
9.1. Estructura básica de la aplicación	73
9.1.1. Clase <code>_basic_object3D</code>	75
9.2. Picking	77
9.3. Selección mediante rectángulo	78
10. Conclusiones y vías futuras	81

Resumen y palabras clave

En el ámbito de las matemáticas, el trabajo fin de grado que presentamos supone la incursión a un campo dentro de la optimización, los teoremas de la alternativa, y sus aplicaciones, principalmente a la propia optimización y finanzas. El trabajo comienza con el teorema de Mazur-Orlicz-König, versión del conocido teorema de Hahn-Banach. Posteriormente, estudiamos el teorema de la alternativa de Gordan, esencial para el resto de resultados. Su primera aplicación se da en la teoría minimax, que nos conduce a resultados clásicos sobre separación de convexos. A continuación, deduciremos otro teorema de la alternativa, el de Farkas, y lo aplicaremos a la programación lineal. Para demostrar los resultados sobre optimización, volvemos al teorema de Gordan, que nos proporcionara los teoremas de Fritz John y Karush-Kuhn-Tucker. Finalmente, nos introducimos en el mundo de las matemáticas financieras obteniendo, gracias al teorema de separación, el primer teorema de asignación de precios para valorar opciones europeas. Concluimos realizando simulaciones de su valor en diferentes casos.

Por su parte, dentro del campo de la informática, se ha realizado un estudio acerca del procesamiento de nubes de puntos. En general, se tienen varias nubes que corresponden a un mismo objeto pero que están tomadas desde distintos puntos de referencia. Esto hace necesario un tratamiento de los conjuntos para conseguir que se encuentren en el mismo sistema de coordenadas y tener así un modelo digital del objeto. Este problema se denomina alineado y para resolverlo se aborda el estudio de dos algoritmos diferentes: ICP y RANSAC. El primero de ellos es un algoritmo genérico que veremos que tiene una gran desventaja, el tiempo de ejecución del mismo. Por ello, intentaremos disminuir todo lo posible este tiempo mediante el uso de descriptores para detectar puntos significativos del modelo y reducir, de ese modo, el conjunto de puntos a los que es necesario aplicar el algoritmo. Por su parte, el algoritmo RANSAC es genérico y sirve para estimar parámetros de un modelo matemático. En nuestro caso, lo usamos para la detección de planos que nos aportaran un número reducido de puntos clave a los que aplicar el proceso de alineado.

Palabras clave: teoremas de la alternativa, teorema de Hahn-Banach, minimax, optimización, matemáticas financieras, alineado, cuaternios, ICP, RANSAC.

Resumen extendido y palabras clave en inglés

In Mathematics, the end of degree project that we present is an incursion into a field included in optimization, the theorems of the alternative, and their applications, mainly to the own optimization and finances.

Many of the theorems of the alternative are just reformulations of convex separation's theorems in certain contexts, so we start this memory with a study on Hahn-Banach's theorem. There are several equivalent versions of this result, collected for example in [7]. That is the reason why many authors talk about Hahn-Banach's theorems. We will focus on one of them, Mazur-Orlicz-König's theorem. Despite its geometrical charge, it is mainly an algebraical result. We will provide a proof of that theorem – the results in this work are self-contained – and we will use it to give a convex version of Gordan's theorem of alternative. Actually, it is an equivalent result due to S. Simons. This version is more general than the original Gordan's result and it will be extremely useful in the next important results.

Then, we will apply the convex Gordan's theorem of the alternative in the minimax theory. A minimax inequality guarantees, under certain hypothesis, that in a two variables function we can substitute $\inf\sup$ for $\sup\inf$. The power of minimax inequalities is clear when we use it – in one that we will deduce from the convex Gordan's theorem – to give classical convex separation's results.

Next, we will deduce from one separation's theorem the Farkas's lemma, which is one of the most known theorem of the alternative. This will allow us to prove, almost immediately, one of the key points of linear programming: the duality theorem. Considering more general optimization theorems, those in which the objective function and the inequalities constraints are differentiables, we will establish, using Gordan's theorem of the alternative, the theorems of Fritz John and Karush-Kuhn-Tucker.

We will conclude with a little foray into the field of financial mathematics. After introducing some of the main concepts, like a derivate security; we will prove, as a consequence of one of the convex separation's theorem, the result known as “First Fundamental Theorem of Asset Pricing”. It can be used in the pricing of European's options in the binomial model. Finally, we have programmed using *Sagemath 2.8*

some examples in order to know how their price varies according to its parameters.

Related to Computer Science, this project consists in a study of 3D digitalization process, which is considered part of Computer Graphics. First, in order to have a digital copy of some desired object, we must obtain some point clouds representing it. They are like three-dimensional photographs. Using this analogy, one photograph does not cover the whole object's surface. That is the reason why we need a few clouds for each one. Unfortunately, the point clouds are not in the same reference system. We need to "modify" them so they match and finally obtain a 3D copy of the object. This modification or treatment have three different processes: alignment, meshing and triangulation. The first one refers to the fact that we mentioned before, the points clouds are not in the same reference system. This step solves this problem. To accomplish this target in a proper way, our point clouds need some overlapping to know exactly how to place them. This situation leads us to the meshing step. Due to overlapping, once we have the shots expressed in the same coordinates, some parts have a higher level of points density. With meshing, we refer to omit "repeated" points in these zones that has been registered several times to get a good alignment. In this moment, we have the object represented by the points. The goal of triangulating is just getting the surface represented by the points. The three steps that we have just mentioned, are very interesting and they deserve a complete study to solve them. However, in this project we will focus in the clouds alignment, also called data registration.

The first algorithm we will study is the Iterative Closest Point (ICP). It is considered a generic algorithm because it can be used to align two point clouds independently of the characteristics of the object. We will talk about this topic lately in this work. However, it is important to highlight that there is no suitable algorithm to align any shots from all type of objects. This section starts with a brief introduction to quaternions. Quaternions are like an extension of complex numbers. In our case, we will use them to represent rotations in three dimensions (a rotation and a translation are needed to align the clouds). There are other ways to represent them such as 3×3 matrixes or Euler angles but quaternions have some advantages. After that, we will discuss how to obtain the matrix and the translation mentioned before by optimizing a distance function. This will give us the steps we must follow in the algorithm. We will also proof the local convergence of this method. The fact that it converges locally forces us to have a pre-alignment step to ensure that the result is correct. Another disadvantage of ICP is the quadratic complexity in time which make it untreatable with large point clouds.

Our next goal will be to reduce the amount of time spent by the algorithm to complete. We will achieve it by reducing the point clouds and just using the most relevant of the model in our method. To detect these points, called key points, we will use the variation of the normal vector to distinguish between planar and sharp areas. It is possible thanks to the scanner used to get the shots. This decrement of the points' quantity will cause a significant reduction in the execution time. Finally, we will study how affects the pre-alignment in the provided results.

The second algorithm we will discuss is the Random Sample Consensus (RANSAC). It is used to get the parameters that fit a mathematical model given by some

observations. It is considered a robust algorithm even in the presence of outliers and is based on an hypothesis-test paradigm. In our case, we will use it to detect planes in our model. The steps to obtain the planes are very simple: select three random points that will represent a plane, count how many points fits that plane (with some tolerance), repeat this process N times and take the one that has the maximum number of points. We will give a formula to determine the number N depending on the probability to be an outlier, the desired probability to find the model and the number of parameters needed to define it. Once we have the planes, we will calculate their intersections in order to have significant corners. Then, we will use the angles between the planes to assign a descriptor to the intersection points and determine which of them match in different point clouds.

To test the proposed methods, we have different models: a clay feet property of the Faculty of Fine Arts of Granada, a part of *La torre de las Gallinas* in the Alhambra and an office. These models have been taken with a Faro Focus 130 laser scanner. It is also necessary to develop an application in which we can code and observe the results of the different algorithms. We will cover this part on the last chapter. This program is written in *C++*, we use *OpenGL* as graphic library and *Qt* for the user interface. During its development, we have applied several concepts related to Computer Graphics. For instance, methods to select or delete a bunch of points or pick one of them. We have also been concerned about the application performance, especially in our case where we use models with lot of vertices.

Keywords: theorems of the alternative, Hahn-Banach's theorem, minimax, optimization, financial mathematics, data registration, quaternions, ICP, RANSAC.

Introducción

El estudio de los teoremas de la alternativa hunde sus raíces en el teorema de separación de convexos de Hahn-Banach. Es más, algunas de sus versiones más conocidas, como el teorema de la alternativa de Gordan o el lema de Farkas, son precursoras de ese resultado fundamental del análisis funcional o la optimización. Y es precisamente ahí, en un contexto de optimización, donde surgen hace casi siglo y medio. Desde entonces han aparecido en gran cantidad, vinculados a problemas de optimización convexa en muchas ocasiones y mediante el uso de técnicas de separación. Además, su aplicabilidad no se ha circunscristo exclusivamente al campo de la optimización sino que ha transcendido dicha área: análisis convexo, análisis funcional, problemas de equilibrio ...

En esta memoria se aborda el estudio de los dos resultados de la alternativa mencionados, el teorema de Gordan y el lema de Farkas, dando incluso una versión más general del primero. Para establecerlos usamos una versión muy simple del teorema de Hahn-Banach y una mejora del mismo. Además, se aplican para establecer una desigualdad minimax que deriva, en particular, en una serie de teoremas de separación de convexos. También se ilustra su aplicabilidad en establecer resultados centrales en la optimización, como es el teorema de dualidad en programación lineal o los teoremas de Fritz John y Karush-Kuhn-Tucker en un contexto diferenciable. Finalmente, dedicamos todo un capítulo de la memoria a usar los teoremas de la alternativa, en una de sus formas equivalentes, para demostrar un resultado importante en el ámbito de las matemáticas financieras, el primer teorema fundamental de valoración de activos financieros. Ello requiere un bagaje previo – conceptos y resultados – que también se recoge en la memoria. Dicho teorema se aplica al caso de ciertos derivados muy populares, las opciones europeas, bajo un modelo binomial y se presentan algunas simulaciones numéricas realizadas con *SageMath* en su versión 2.8.

Centrándonos en la parte informática, el trabajo que se expone a continuación puede considerarse como una introducción al mundo del digitalizado 3D. Es una técnica que se engloba dentro del campo de la Informática Gráfica y que se aplica a una gran cantidad de sectores: diseño industrial, arte, cine, medicina, etc. Por ejemplo, en el mundo artístico se emplea para la obtención de un modelo digital de la obra con el fin de tener una copia digital de la misma u obtener más información que puede ser de utilidad, como en los procesos de restauración, sin necesidad de tener el original. También podemos poner un ejemplo de su uso en diseño industrial donde el modelo obtenido se puede utilizar para realizar simulaciones con el

objetivo de detectar las partes más susceptibles a sufrir fallos.

La obtención de un modelo detallado y completo es un proceso complicado y ampliamente estudiado. En primer lugar, es importante el escáner que utilicemos para realizar dicho proceso. En la actualidad existen una gran cantidad de modelos que utilizan diferentes tecnologías para obtener cada una de las tomas necesarias aunque la mayoría de ellos suelen darnos como resultado una nube de puntos que define la muestra. Otras técnicas como la fotogrametría también tienen como objetivo la obtención de un modelo 3D pero a partir de fotografías. Nosotros estudiaremos mecanismos basados en nubes de puntos. En cuanto a las técnicas utilizadas para la obtención estas nubes de puntos hay dos tipos: por contacto o sin contacto. La más utilizada es la segunda de ellas que a su vez puede utilizar una tecnología láser o mediante luz estructurada. Entre otros aspectos a tener en cuenta encontramos la resolución del escáner, su precisión, las propiedades del objeto, etc. No es el objetivo de este trabajo ver las tecnologías disponibles en el digitalizado ni el propio proceso y sus consecuencias en el resultado final obtenido pero es importante conocer las alternativas disponibles. Si hay que tener en cuenta que algunos escáneres nos pueden aportar información adicional acerca de los puntos lo que puede ser de utilidad posteriormente. Una vez tenemos las nubes de punto, podemos distinguir tres etapas hasta llegar al modelo digital:

- Alinear: cuando se toman las muestras del objeto a digitalizar, es necesario mover el escáner con el que se realizan las medidas o el propio objeto. Esto conlleva que diferentes tomas van a estar en diferentes sistemas de coordenadas. Esto se podría solucionar, por ejemplo, mediante un sistema GPS que guardara las posiciones para posteriormente realizar las transformaciones correspondientes en las muestras. El error de GPS puede ser del orden de varios metros y no funciona bien en interiores. Por otra parte hay otro tipo de mecanismos que puede merecer la pena comentar: utilizar dianas en la escena; sensores iniciales en el escáner. En cualquier caso estos mecanismos no están siempre disponibles. Por ello, esta etapa, tiene como objetivo que todas las tomas estén respecto al mismo sistema de coordenadas. Otro aspecto a destacar es si estas tomas son rígidas o no, es decir, si no hay deformaciones en las mismas. Nosotros en este trabajo nos centraremos en el caso rígido.
- Fusionar: finalizada la etapa anterior debemos hacer que todas las nubes de puntos que tenemos alineadas se transformen en una sola. Más concretamente, durante la obtención de las muestras es necesario que estas se solapen, como veremos posteriormente, lo que hace que haya zonas con gran cantidad de puntos, muchos de ellos “repetidos”. De este modo, debemos detectar dichos puntos para que la densidad en todo el modelo sea uniforme. También hay que tener en cuenta que es posible que haya zonas que por falta de previsión o por imposibilidad física no se hayan podido obtener una muestra. En esta etapa también se deberían arreglar dichas faltas de información.
- Triangular: hasta ahora hemos trabajado con nubes de puntos pero el modelo es una superficie. Con esta etapa final se pretende obtener los triángulos que definen dicha superficie y así completar el proceso de digitalizado. Una alternativa sería obtener una función implícita que defina la superficie.

Las etapas que acabamos de mencionar no deben darse obligatoriamente en ese orden. Podríamos empezar calculando los triángulos que define cada nube de

puntos y posteriormente alinear y fusionar. Sin embargo, el orden descrito es el más habitual. Durante el proceso también se suele tener en cuenta la densidad de puntos, la posible existencia de ruido, de valores atípicos o del error que tiene el propio escáner durante la toma de medidas aunque no serán aspectos a los que nosotros les demos demasiada importancia.

Una idea importante a destacar es que en el digitalizado 3D no hay un algoritmo que sea capaz de reconstruir todo tipo de objetos. En este trabajo se estudiarán algunos algoritmos genéricos pero en la práctica existen métodos específicos para entornos urbanos, arquitectónicos, modelos curvilíneos, suaves o con aristas, etc. Más concretamente, nos centraremos en la primera de las etapas del proceso, el alineado. Dentro de este campo veremos dos de los algoritmos más usados: ICP y RANSAC. El primero de ellos es un algoritmo que se usa para cualquier modelo pero que veremos que tiene una gran desventaja: el tiempo de ejecución del mismo. Por ello, intentaremos disminuir todo lo posible este tiempo mediante el uso de descriptores para detectar puntos significativos del modelo y reducir de ese modo el conjunto de puntos a los que es necesario aplicar el algoritmo. Por su parte, el algoritmo RANSAC es más específico que el anterior y sirve para estimar parámetros de un modelo matemático. En nuestro caso, lo usaremos para la detección de planos que nos aportaran un número reducido de puntos clave a los que aplicar el proceso de alineado. Estos algoritmos se han programado dentro un banco de pruebas que nos permite realizar los algoritmos paso a paso con el fin de poder observar mejor el comportamiento de los mismos.

En definitiva, por un lado, en esta memoria se plasma tanto el carácter convexo-funcional de los teoremas de la alternativa como su aplicabilidad a campos tan diversos como la optimización, el análisis convexo y las matemáticas financieras. Por la parte informática, se ha realizado un análisis de dos procedimientos habituales para la alineación de dos nubes de puntos, proceso que se engloba dentro de la digitalización 3D. Señalamos finalmente que las referencias usadas en la elaboración de este memoria aparecen recogidas en el capítulo de Bibliografía. No obstante, los textos de [1], [2] y [8] han sido los esenciales para la parte de matemáticas financieras, mientras que [10], [12], [14] y [16] han sido de gran utilidad en el tema de digitalizado.

Objetivos

Los objetivos inicialmente previstos en la propuesta de TFG en matemáticas fueron:

- Realizar una recopilación de algunos teoremas de la alternativa.
- Teorema de dualidad en programación lineal.
- Teoremas de Karush-Kuhn-Tucker y Fritz John para programación convexa.
- Aplicación a finanzas: teorema fundamental de valoración de activos financieros en mercados finitos.

Sin embargo, nuestro tratamiento final ha sido algo más ambicioso, pues hemos incluido todo un capítulo de aplicaciones de los teoremas de la alternativa a la teoría minimax y a la separación de convexos. Además, en lugar de considerar los teoremas de Karush-Kuhn-Tucker y Fritz John en un ambiente convexo, los hemos establecido en un contexto no lineal y diferenciable. La idea que nos ha llevado a ello ha sido aumentar el número y tipología de aplicaciones de los teoremas de la alternativa, mostrando su versatilidad en diversos campos.

Dentro de la parte informática se pretendía:

- Realizar una breve incursión en el mundo de la digitalización 3D, conociendo a rasgos generales el proceso empleado en la obtención de un modelo.
- Estudio de la información geométrica del modelo obtenido.
- Uso de dicha información para proponer una optimización de los algoritmos empleados en el proceso.
- Realización de un programa para evaluar los resultados obtenidos.

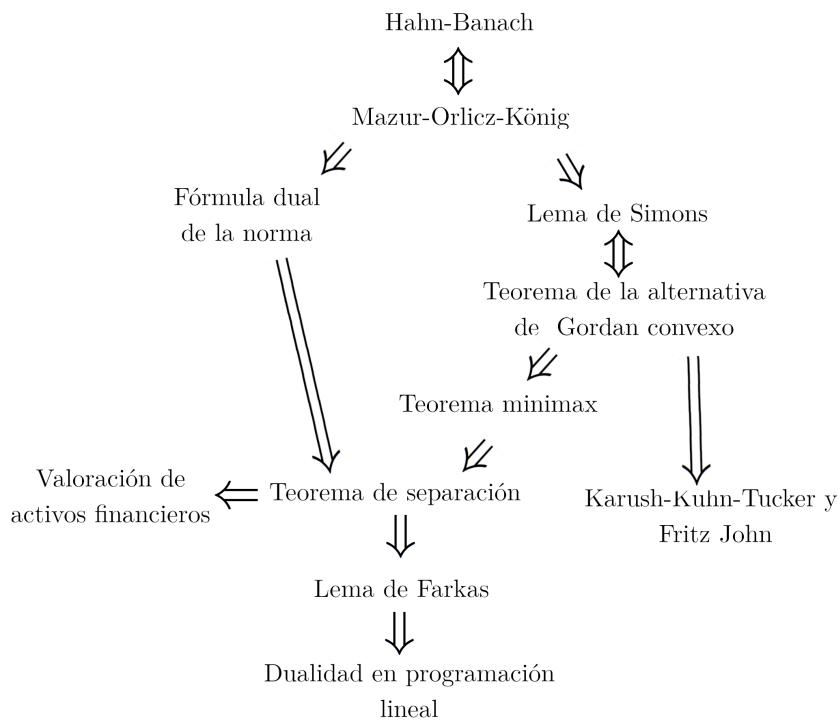
Destacamos que también se han alcanzado los objetivos propuestos en este caso. Inicialmente, la propuesta de TFG no contenía ninguna especificación acerca de los algoritmos o técnicas a usar debido a la amplia cantidad de métodos existentes así como la complejidad del proceso completo de digitalizado. Así, el primer punto ha servido como fase previa para el conocimiento del tema a trabajar para poder concretar un objetivo. Tras ello, se decidió centrar el estudio en la etapa de alineado tal, como ya se ha comentado anteriormente, y orientar de ese modo, los puntos segundo y tercero en ese ámbito. Finalmente, el cuarto punto se ha enfocado como un banco de pruebas para comprobar los resultados obtenidos en los puntos anteriores y no como una aplicación para uso de terceros. Esto ha conllevado el uso de

nuevas herramientas de desarrollo así como profundizar en conceptos clave de la Informática Gráfica. De este modo, se han alcanzado estos objetivos de una manera colateral.

Desarrollo

Matemáticas

El proceso seguido en el desarrollo del TFG ha sido, por un lado, recopilar material sobre el tema y analizarlo, y por otro, darle estructura totalmente auto-contenida, elaborando los diversos contenidos de forma jerarquizada en el sentido de que se deducen de los anteriores. A modo de esquema, los resultados se han estructurado atendiendo al siguiente esquema donde además se recoge la relación entre ellos:



Como puede observarse, las técnicas son de carácter convexo y analítico funcional.

Informática

Tras la concreción del tema a resolver dentro del ámbito del digitalizado 3D, se decidió empezar a trabajar en uno de los algoritmos más conocidos para ello, el *Iterative Closest Point* (ICP). También, al comenzar el trabajo, se decidió la manera de realizar las pruebas necesarias. Se optó por la realización de un banco de pruebas utilizando *OpenGL* (versión 4.6) como biblioteca gráfica, *C++* como lenguaje de programación así como *Qt* (versión 5.9.1) para la interfaz gráfica. El IDE, por lo tanto, ha sido *Qt Creator 4.3.1*. Este banco de pruebas debe proporcionar herramientas básicas para la manipulación de las nubes de puntos con las que estamos trabajando así como otras funcionalidades necesarias para el desarrollo de los distintos algoritmos. En este caso se hizo necesaria solventar problemas como: abrir y guardar archivos, manipulación de la matriz de vista, selección de un conjunto de puntos y borrado de los mismos, selección de puntos aislados, etc.

Volviendo al primer algoritmo a estudiar, se comprobó que era necesario un conocimiento previo acerca de los cuaternios, tanto para la demostración de la convergencia del método como para su implementación. Una vez completado, se procedió a la elaboración de distintas pruebas. Destacar que también se hizo necesario el cálculo de valores propios de una matriz por lo que se optó por usar la biblioteca *Eigen* (en su versión 3.3.7). Posteriormente, se planteó la posibilidad de incluir mejoras al método usando la variación de las normales para obtener información acerca de la geometría del modelo. Por ello, fue necesaria la inclusión de esta opción dentro del banco de pruebas así como un algoritmo de simplificado de la nube de puntos tal y como se explicará en el desarrollo del trabajo.

Llegados a este punto, se planteó la posibilidad de incluir otra serie de mejoras al procedimiento o de estudiar otros algoritmos diferentes. Se siguió la segunda opción con el fin de tener una visión general más general del problema. Por ello, en último lugar aparece un estudio acerca del algoritmo *Random Sample Consensus* (RANSAC), partiendo de la explicación de los pasos del propio algoritmo pasando por razonamientos probabilísticos para asegurar la obtención de buenos resultados y finalmente la realización de pruebas del mismo.

Parte I

Teoremas de la alternativa, optimización convexa y valoración de activos financieros

Parte II

Procesamiento de nubes de puntos generadas por escáner láser

Iterative Closest Point (ICP)

En esta sección explicaremos el primer algoritmo denominado *Iterative Closest Point* (ICP), o Iteración del Punto más Cercano. Este algoritmo se utiliza en la etapa de alineación y su finalidad es que todas las tomas de datos estén referenciadas respecto al mismo sistemas de coordenadas. Este proceso también se suele denominar *registro de datos* o *data registration*. Veremos el método inicial propuesto en 1992 por Paul J.Besl and Neil D.MacKay [12]. Como veremos, este es un algoritmo complejo en tiempo y por ello, se tarda bastante en obtener los resultados deseados. Por ello, una vez estudiado el algoritmo se propondrán unas pequeñas mejoras del mismo que consistirán, básicamente, en el uso de las normales en cada punto para acelerar el proceso.

Antes de explicar el algoritmo necesitamos unas nociones previas acerca de los *cuaternios*. Los utilizaremos para la representación de las rotaciones en tres dimensiones, frente a la forma tradicional de representación matricial, y serán de gran ayuda a la hora de demostrar la convergencia del algoritmo. Tanto para la explicación de los cuaternios como su uso en optimización se ha tomado como referencia [10].

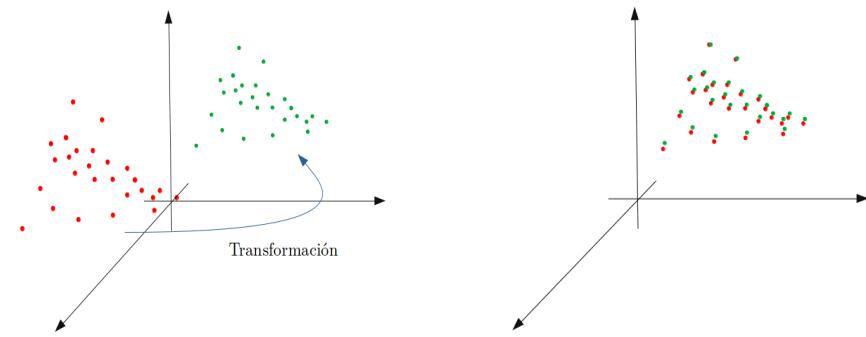


Figura 6.1: Antes y después de alinear dos nubes de puntos.

6.1. Cuaternios

Su desarrollo se atribuye a W.R Hamilton en 1843. Son una extensión de los números complejos. Mientras que estos últimos solo incluyen la unidad imaginaria

\mathbf{i} , los cuaternios añaden la unidades imaginarias $\mathbf{i}, \mathbf{j}, \mathbf{k}$ tales que:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1.$$

Así, el conjunto de los cuaternios se define como:

$$\{q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} : q_0, q_1, q_2, q_3 \in \mathbb{R}\}.$$

Podemos ver a los cuaternios como la suma de un escalar $q_0 \in \mathbb{R}$ y de un vector de \mathbb{R}^3 que llamamos $\mathbf{q} = (q_1, q_2, q_3)$ donde la base usual de \mathbb{R}^3 viene dada por los vectores unitarios $\mathbf{i} = (1, 0, 0)$, $\mathbf{j} = (0, 1, 0)$ y $\mathbf{k} = (0, 0, 1)$. Notaremos $q = q_0 + \mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$. Si la parte escalar es 0, es decir, $q_0 = 0$ hablamos de un *cuaterniono puro*. Por otro lado, a modo de notación podemos expresar q como vectores de cuatro dimensiones, es decir, si q es de la forma mencionada podemos pensar que $q = (q_0, q_1, q_2, q_3) \in \mathbb{R}^4$.

6.1.1. Suma y producto

La suma de dos cuaternios se realiza componente a componente. Por ello, dados los cuaternios

$$\begin{aligned} q &= q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} = q_0 + \mathbf{q} \\ p &= p_0 + p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k} = p_0 + \mathbf{p}, \end{aligned}$$

definimos su suma de la siguiente manera:

$$q + p = (q_0 + p_0) + (q_1 + p_1)\mathbf{i} + (q_2 + p_2)\mathbf{j} + (q_3 + p_3)\mathbf{k}.$$

Es directo comprobar que la suma es asociativa y commutativa.

El producto, por su parte, sigue una serie de reglas introducidas por Hamilton y que las presentamos a continuación:

$$\begin{aligned} \mathbf{i}^2 &= \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1. \\ \mathbf{ij} &= \mathbf{k} = -\mathbf{ji}. \\ \mathbf{jk} &= \mathbf{i} = -\mathbf{kj}. \\ \mathbf{ki} &= \mathbf{j} = -\mathbf{ik}. \end{aligned}$$

Por ello, el producto de dos cuaternios viene dado por:

$$\begin{aligned} pq &= (p_0 + p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k})(q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}) \\ &= p_0q_0 - (p_1q_1 + p_2q_2 + p_3q_3) + p_0(q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}) \\ &\quad + q_0(p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k}) + (p_2q_3 - p_3q_2)\mathbf{i} + (p_3q_1 - p_1q_3)\mathbf{j} \\ &\quad + (p_1q_2 - p_2q_1)\mathbf{k}. \end{aligned}$$

Podemos simplificar esta expresión usando el producto escalar y vectorial de \mathbb{R}^3 quedando la expresión más concisa:

$$pq = p_0q_0 - \langle \mathbf{p}, \mathbf{q} \rangle + p_0\mathbf{q} + q_0\mathbf{p} + \mathbf{p} \times \mathbf{q}.$$

No es difícil comprobar que el producto es asociativo y distributivo respecto a la suma pero no commutativo. La no commutatividad se puede observar del hecho de que

el producto vectorial no es commutativo. Es importante notar que los cuaternios son cerrados bajo la suma y el producto definidos anteriormente. En el caso del producto, obtenemos otro cuaternion con parte escalar $p_0 q_0 - \langle \mathbf{p}, \mathbf{q} \rangle$ y vector $p_0 \mathbf{q} + q_0 \mathbf{p} + \mathbf{p} \times \mathbf{q}$.

Este conjunto con las operaciones de suma y producto forman un anillo no commutativo. El elemento neutro para la suma sería un cuaternion donde $q_0 = q_1 = q_2 = q_3 = 0$ y dado un elemento $q = q_0 + \mathbf{q}$ su opuesto sería $-q = -q_0 - \mathbf{q}$. Por su parte, el elemento neutro para el producto sería un cuaternion con parte real 1 y vector $\mathbf{0}$.

6.1.2. Conjugado, norma e inverso

Sea el cuaternion dado por $q = q_0 + q_1 \mathbf{i} + q_2 \mathbf{j} + q_3 \mathbf{k}$. Notamos a su *conjugado* como q^* y que se define como:

$$q^* = q_0 - \mathbf{q} = q_0 - q_1 \mathbf{i} - q_2 \mathbf{j} - q_3 \mathbf{k}.$$

De la definición obtenemos las siguientes propiedades básicas del manejo de cuaternios:

$$\begin{aligned} (q^*)^* &= q - (-\mathbf{q}) = q. \\ q + q^* &= q_0. \\ qq^* &= q^*q = (q_0 + \mathbf{q})(q_0 + \mathbf{q}) \\ &= q_0 q_0 - \langle -\mathbf{q}, \mathbf{q} \rangle + q_0 \mathbf{q} + (-\mathbf{q}) q_0 + (-\mathbf{q}) \times \mathbf{q} \\ &= q_0^2 + \langle \mathbf{q}, \mathbf{q} \rangle \\ &= q_0^2 + q_1^2 + q_2^2 + q_3^2 \\ &= q_0^2 + \|\mathbf{q}\|. \end{aligned}$$

En la última expresión $\|\cdot\|$ representa la norma usual de \mathbb{R}^3 . Podemos probar que dados dos cuaternion p, q se cumple que $(pq)^* = q^* p^*$. Efectivamente:

$$\begin{aligned} (pq)^* &= p_0 q_0 - \langle \mathbf{p}, \mathbf{q} \rangle - p_0 \mathbf{q} - q_0 \mathbf{p} - \mathbf{p} \times \mathbf{q} \\ &= p_0 q_0 - \langle \mathbf{p}, \mathbf{q} \rangle - p_0 \mathbf{q} - q_0 \mathbf{p} + \mathbf{q} \times \mathbf{p} \\ &= q_0 p_0 - \langle -\mathbf{q}, -\mathbf{p} \rangle + q_0 (-\mathbf{p}) + p_0 (-\mathbf{q}) + (-\mathbf{q}) \times (-\mathbf{p}) \\ &= (q_0 - \mathbf{q})(p_0 - \mathbf{p}) \\ &= q^* p^*. \end{aligned}$$

La *norma* de un cuaternion se nota como $|q|$ y se define como $|q| = \sqrt{q^* q}$. Usando la propiedad anterior de que el conjugado del producto es el producto de los conjugados cambiados de orden llegamos a:

$$\begin{aligned} |pq|^2 &= (pq)^*(pq) \\ &= q^* p^* pq \\ &= q^* |p|^2 q \\ &= |p|^2 q^* q \\ &= |p|^2 |q|^2. \end{aligned}$$

Por su parte, el *inverso* de un cuaternion se define como:

$$q^{-1} = \frac{q^*}{|q|^2}.$$

Se cumple que:

$$qq^{-1} = q^{-1}q = \frac{q^*}{|q|^2}q = \frac{|q|^2}{|q|^2} = 1.$$

En el caso de que q sea unitario (tenga norma uno) tenemos que $q^{-1} = q^*$. Notar que los cuaternios unitarios forman un grupo con operación interna el producto.

6.1.3. Cuaternios y rotaciones

El grupo de rotaciones en \mathbb{R}^3 alrededor de un eje que pasa por el origen, y que lo notamos como $SO(3)$, está formado por las matrices reales 3×3 que son ortogonales (su inversa coincide con la traspuesta) y tienen determinante igual a 1.

Nos surge entonces la siguiente pregunta: ¿por qué usar los cuaternios para representar dichas rotaciones? En primer lugar, el uso de matrices resulta redundante. Las rotaciones en \mathbb{R}^3 tienen básicamente tres grados de libertad: dos para la representación del eje (dado en coordenadas esféricas, por ejemplo) y uno para el ángulo de giro. Por ello, al usar matrices estamos usando demasiados elementos. Con el uso de los cuaternios seguimos teniendo redundancia pero solo por un valor más. Además, a partir de un eje de rotación y un ángulo de giro se puede construir fácilmente el cuaternion asociado, y del mismo modo, obtener el eje y ángulo a partir del cuaternion, mientras que en matrices dicha relación no es tan clara. También reducimos el número de operaciones necesarias a la hora de componer dos rotaciones. En la multiplicación de matrices se necesitan 27 multiplicaciones y 18 sumas y en el producto de cuaternios (veremos que esta es la operación de composición) 16 multiplicaciones y 20 sumas. Por otro lado, podemos usar los *ángulos de Euler* para identificar las rotaciones que utiliza el número de valores mínimo. Sin embargo, con este sistema podemos llegar a una situación no deseada que se conoce como *gimbal lock* o *bloqueo del cardán* [11]. Este fenómeno se produce cuando los tres ejes están en el mismo plano y nos produce la pérdida de uno de los ejes de giro. Por estas razones, entre otras, los cuaternios se conocen generalmente como la “mejor” manera para representar las rotaciones en el espacio. Más específicamente, la representación de las rotaciones en tres dimensiones se hace mediante cuaternios unitarios. Veamos el proceso:

Sea q un cuaternion unitario de la forma $q = q_0 + \mathbf{q}$. Al tener norma uno tenemos que $|q|^2 = q_0^2 + \|\mathbf{q}\|^2 = 1$. Así, como $(q_0, \|\mathbf{q}\|) \in \mathbb{S}^1$, existe un único ángulo $\theta \in [0, 2\pi)$ que cumple:

$$\begin{cases} \cos^2 \theta = q_0^2 \\ \sin^2 \theta = \|\mathbf{q}\|^2, \end{cases}$$

y podemos escribir ahora q en términos de θ y \mathbf{u} con $\mathbf{u} = \mathbf{q}/\|\mathbf{q}\|$ de la siguiente manera: $q = \cos \theta + \mathbf{u} \sin \theta$.

Dado el cuaternion unitario q , definimos la aplicación L_q dada por:

$$\begin{aligned} L_q : \mathbb{R}^3 &\longrightarrow \mathbb{R}^3 \\ \mathbf{v} &\longmapsto L_q(\mathbf{v}) = q\mathbf{v}q^* = (q_0^2 - \|\mathbf{q}\|^2)\mathbf{v} + 2\langle \mathbf{q}, \mathbf{v} \rangle \mathbf{q} + 2q_0(\mathbf{q} \times \mathbf{v}) \end{aligned}$$

La aplicación L_q está bien definida y no tenemos problema en multiplicar un vector por un cuaternion, ya que podemos ver un vector como un cuaternion puro. Hacemos ahora dos observaciones interesantes que nos hacen pensar, en principio, que la aplicación L_q actúa como una rotación con eje \mathbf{q} .

Observación 6.1. *La aplicación L_q no cambia la norma de $\mathbf{v} \in \mathbb{R}^3$.*

En efecto:

$$\|L_q(\mathbf{v})\| = \|q\mathbf{v}q^*\| = |q| \|\mathbf{v}\| |q^*| = \|\mathbf{v}\|.$$

Observación 6.2. *Si $\mathbf{v} = k\mathbf{q}$ con $k \in \mathbb{R}$ entonces la dirección de \mathbf{v} no cambia al aplicarle L_q .*

Usando la definición de L_q y al ser q unitario llegamos a:

$$\begin{aligned} L_q(\mathbf{v}) &= q\mathbf{v}q^* \\ &= (q_0^2 - \|\mathbf{q}\|^2)(k\mathbf{q}) + 2\langle \mathbf{q}, k\mathbf{q} \rangle \mathbf{q} + 2q_0(\mathbf{q} \times k\mathbf{q}) \\ &= k(q_0^2 - \|\mathbf{q}\|^2)\mathbf{q} + 2k\langle \mathbf{q}, \mathbf{q} \rangle \mathbf{q} + 2kq_0(\mathbf{q} \times \mathbf{q}) \\ &= k(q_0^2 - \|\mathbf{q}\|^2)\mathbf{q} + 2k\|\mathbf{q}\|^2 \mathbf{q} \\ &= k(q_0^2 + \|\mathbf{q}\|^2)\mathbf{q} \\ &= k\mathbf{q}. \end{aligned}$$

Antes de formalizar la “intuición” que nos han dado las observaciones anteriores necesitamos otra que es un poco más técnica.

Observación 6.3. *La aplicación L_q es lineal, es decir, dados $a_1, a_2 \in \mathbb{R}$ y $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^3$ se tiene que $L_q(a_1\mathbf{v}_1 + a_2\mathbf{v}_2) = a_1L_q(\mathbf{v}_1) + a_2L_q(\mathbf{v}_2)$.*

Se obtiene por la distributividad del producto:

$$\begin{aligned} L_q(a_1\mathbf{v}_1 + a_2\mathbf{v}_2) &= q(a_1\mathbf{v}_1 + a_2\mathbf{v}_2)q^* \\ &= q(a_1\mathbf{v}_1q^* + a_2\mathbf{v}_2q^*) \\ &= qa_1\mathbf{v}_1q^* + qa_2\mathbf{v}_2q^* \\ &= a_1q\mathbf{v}_1q^* + a_2q\mathbf{v}_2q^* \\ &= a_1L_q(\mathbf{v}_1) + a_2L_q(\mathbf{v}_2). \end{aligned}$$

El teorema que exponemos a continuación es clave y nos indica que dado un vector y ángulo podemos construir un cuaternion q tal que L_q representa la rotación deseada.

Teorema 6.1. Para cualquier cuaternion $q = q_0 + \mathbf{q} = \cos \theta/2 + \mathbf{u} \operatorname{sen} \theta/2$ y para cualquier vector $\mathbf{v} \in \mathbb{R}^3$ tenemos que $L_q(\mathbf{v})$ es equivalente a la rotación con eje de giro \mathbf{u} y ángulo θ .

Demuestração. Tomamos $\mathbf{v} \in \mathbb{R}^3$ y lo expresamos en términos de \mathbf{q} y $\{\mathbf{q}\}^\perp$. De este modo, $\mathbf{v} = \mathbf{a} + \mathbf{n}$ donde \mathbf{a} es la componente respectiva a \mathbf{q} y \mathbf{n} la de $\{\mathbf{q}\}^\perp$. Bajo L_q , \mathbf{a} se queda invariante por la observación 6.2 mientras que \mathbf{n} rota un ángulo θ alrededor de \mathbf{q} . En efecto:

$$\begin{aligned} L_q(\mathbf{n}) &= (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2\langle \mathbf{q}, \mathbf{n} \rangle \mathbf{q} + 2q_0(\mathbf{q} \times \mathbf{n}) \\ &= (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2q_0(\mathbf{q} \times \mathbf{n}) \\ &= (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2q_0 \|\mathbf{q}\| (\mathbf{u} \times \mathbf{n}), \end{aligned}$$

donde la última igualdad se debe a que $\mathbf{u} = \mathbf{q}/\|\mathbf{q}\|$. Notamos $\mathbf{n}_0 = \mathbf{u} \times \mathbf{n}$. Es claro por ello que \mathbf{n}_0 es perpendicular tanto a \mathbf{u} (y por ello también a \mathbf{q}) y a \mathbf{n} . Obtenemos:

$$L_q(\mathbf{n}) = (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2q_0 \|\mathbf{q}\| \mathbf{n}_0. \quad (6.1)$$

Tanto \mathbf{n}_0 como \mathbf{n} tienen la misma longitud:

$$\|\mathbf{n}_0\| = \|\mathbf{n} \times \mathbf{u}\| = \|\mathbf{u} \times \mathbf{n}\| = \|\mathbf{n}\| \|\mathbf{u}\| \operatorname{sen}(\pi/2) = \|\mathbf{n}\|.$$

Escribimos 6.1 es términos de θ :

$$\begin{aligned} L_q(\mathbf{n}) &= (\cos^2 \frac{\theta}{2} - \operatorname{sen}^2 \frac{\theta}{2})\mathbf{n} + (2 \cos^2 \frac{\theta}{2} \operatorname{sen}^2 \frac{\theta}{2})\mathbf{n}_0 \\ &= \cos \theta \mathbf{n} + \operatorname{sen} \theta \mathbf{n}_0. \end{aligned}$$

Las últimas igualdades se han obtenido mediante las fórmulas del seno y coseno del ángulo doble. Vemos que el vector resultante es la rotación de \mathbf{n} un ángulo θ en el plano definido por \mathbf{n} y \mathbf{n}_0 . Claramente el vector resultante es perpendicular al eje de rotación ya que \mathbf{n} y \mathbf{n}_0 pertenecen a $\{\mathbf{q}\}^\perp$. La demostración termina usando la linealidad que nos aporta la observación 3. \blacksquare

En resumen, dados $\mathbf{q} \in \mathbb{R}^3$ y $\theta \in \mathbb{R}$ representamos la rotación de ángulo de giro θ alrededor del eje \mathbf{q} por $q = \cos \frac{\theta}{2} + \frac{\mathbf{q}}{\|\mathbf{q}\|} \operatorname{sen} \frac{\theta}{2}$ y la aplicamos a un vector $\mathbf{u} \in \mathbb{R}^3$ mediante $L_q(\mathbf{u}) = \mathbf{q} \mathbf{u} q^*$.

Finalmente, exponemos la siguiente observación que nos indica la relación de la composición de rotaciones con su representación mediante cuaternios.

Observación 6.4. Sean p, q dos cuaternios unitarios, entonces $L_q \circ L_p = L_{qp}$.

Como p y q son unitarios entonces pq también lo es. Notamos $L_p(\mathbf{u}) = \mathbf{v}$ y $L_q(\mathbf{v}) = \mathbf{w}$. Obtenemos:

$$\begin{aligned} \mathbf{w} &= L_q(\mathbf{v}) = q \mathbf{v} q^* \\ &= q(p u p^*) q^* \\ &= (qp) \mathbf{u} (qp)^* \\ &= L_{qp}(\mathbf{u}). \end{aligned}$$

De este modo, hemos obtenido que la composición de rotaciones representadas en cuaternios se obtiene mediante la multiplicación de los cuaternios que representa cada una de las rotaciones por separado.

6.2. Cuaternios y optimización

Sea $P = \{\mathbf{p}_i\}$ un conjunto con N_p puntos que queremos alinear tomando como sistema de referencia otro conjunto $X = \{\mathbf{x}_i\}$ de tamaño N_x . Suponemos que $N_p = N_x$ y que cada $\mathbf{p}_i \in P$ corresponde con $\mathbf{x}_j \in X$ siempre y cuando $i = j$. Esta correspondencia hace referencia a que conocemos cómo se relacionan los puntos de ambos conjuntos y por ello p_i es el punto x_i pero en un sistema de referencia diferente. Aclararemos esta relación en apartados siguientes. Suponer que ambos conjuntos de puntos tienen el mismo tamaño puede parecer bastante restrictivo en un primer momento pero posteriormente veremos que no es el caso. De este modo, el problema consiste en encontrar una rotación R y una traslación \mathbf{b} que nos proporcionen las distancias más pequeñas entre los puntos. Así, la función objetivo a ser minimizada es:

$$F(R, \mathbf{b}) = \sum_{i=1}^{N_p} \|R\mathbf{p}_i + \mathbf{b} - \mathbf{x}_i\|^2. \quad (6.2)$$

En primer lugar, obtenemos los centroides de ambos conjuntos de puntos. Recorriendo que que $N_p = N_x$, los notamos como:

$$\bar{\mathbf{p}} = \frac{1}{N_p} \sum_{i=1}^{N_p} \mathbf{p}_i, \quad \bar{\mathbf{x}} = \frac{1}{N_p} \sum_{i=1}^{N_p} \mathbf{x}_i.$$

A continuación, obtenemos las coordenadas relativas a los centroides:

$$\mathbf{p}'_i = \mathbf{p}_i - \bar{\mathbf{p}}, \quad \mathbf{x}'_i = \mathbf{x}_i - \bar{\mathbf{x}}.$$

Notamos que se cumple que:

$$\begin{aligned} \sum_{i=1}^{N_p} \mathbf{p}'_i &= \sum_{i=1}^{N_p} (\mathbf{p}_i - \bar{\mathbf{p}}) = \sum_{i=1}^{N_p} \mathbf{p}_i - N_p \bar{\mathbf{p}} = \sum_{i=1}^{N_p} \mathbf{p}_i - N_p \frac{1}{N_p} \sum_{i=1}^{N_p} \mathbf{p}_i = 0, \\ \sum_{i=1}^{N_p} \mathbf{x}'_i &= \sum_{i=1}^{N_p} (\mathbf{x}_i - \bar{\mathbf{x}}) = \sum_{i=1}^{N_p} \mathbf{x}_i - N_p \bar{\mathbf{x}} = \sum_{i=1}^{N_p} \mathbf{x}_i - N_p \frac{1}{N_p} \sum_{i=1}^{N_p} \mathbf{x}_i = 0. \end{aligned} \quad (6.3)$$

Expresando ahora (6.2) en términos de \mathbf{p}_i , \mathbf{x}_i , $\bar{\mathbf{p}}$ y $\bar{\mathbf{x}}$:

$$\begin{aligned} \sum_{i=1}^{N_p} \|R\mathbf{p}_i + \mathbf{b} - \mathbf{x}_i\|^2 &= \sum_{i=1}^{N_p} \|R(\mathbf{p}'_i + \bar{\mathbf{p}}) + \mathbf{b} - (\mathbf{x}'_i + \bar{\mathbf{x}})\|^2 \\ &= \sum_{i=1}^{N_p} \|R\mathbf{p}'_i - \mathbf{x}'_i + R\bar{\mathbf{p}} + \mathbf{b} - \bar{\mathbf{x}}\|^2 \\ &= \sum_{i=1}^{N_p} \langle R\mathbf{p}'_i - \mathbf{x}'_i + R\bar{\mathbf{p}} + \mathbf{b} - \bar{\mathbf{x}}, R\mathbf{p}'_i - \mathbf{x}'_i + R\bar{\mathbf{p}} + \mathbf{b} - \bar{\mathbf{x}} \rangle. \end{aligned}$$

Para simplificar la escritura, llamamos $a_i = R\mathbf{p}'_i - \mathbf{x}'_i$ y $c = R\bar{\mathbf{p}} + \mathbf{b} - \bar{\mathbf{x}}$. Como el producto escalar es bilineal obtenemos:

$$\begin{aligned} \sum_{i=1}^{N_p} \langle a_i + c, a_i + c \rangle &= \sum_{i=1}^{N_p} \langle a_i + c, a_i \rangle + \sum_{i=1}^{N_p} \langle a_i + c, c \rangle \\ &= \sum_{i=1}^{N_p} \langle a_i, a_i \rangle + \sum_{i=1}^{N_p} \langle c, a_i \rangle + \sum_{i=1}^{N_p} \langle a_i, c \rangle + \sum_{i=1}^{N_p} \langle c, c \rangle \\ &= \sum_{i=1}^{N_p} \|a_i\|^2 + 2 \sum_{i=1}^{N_p} \langle a_i, c \rangle + \sum_{i=1}^{N_p} \|c\|^2 \\ &= \sum_{i=1}^{N_p} \|a_i\|^2 + 2 \sum_{i=1}^{N_p} \langle a_i, c \rangle + N_p \|c\|^2 \\ &= \sum_{i=1}^{N_p} \|a_i\|^2 + 2 \langle \sum_{i=1}^{N_p} a_i, c \rangle + N_p \|c\|^2. \end{aligned}$$

Si sustituimos a_i por su valor en el término central y usamos (6.3):

$$\sum_{i=1}^{N_p} a_i = \sum_{i=1}^{N_p} (R\mathbf{p}'_i - \mathbf{x}'_i) = \sum_{i=1}^{N_p} R\mathbf{p}'_i - \sum_{i=1}^{N_p} \mathbf{x}'_i = 0.$$

Por lo tanto:

$$\sum_{i=1}^{N_p} \|R\mathbf{p}_i + \mathbf{b} - \mathbf{x}_i\|^2 = \sum_{i=1}^{N_p} \|R\mathbf{p}'_i - \mathbf{x}'_i\|^2 + N_p \|R\bar{\mathbf{p}} + \mathbf{b} - \bar{\mathbf{x}}\|^2.$$

La translación \mathbf{b} buscada debería hacer 0 el segundo término obteniendo entonces:

$$\mathbf{b} = R\bar{\mathbf{p}} + \bar{\mathbf{x}}.$$

Ahora, para obtener la mejor rotación utilizamos el primero término. Reescribiendo la misma llegamos a:

$$\begin{aligned}
\sum_{i=1}^{N_p} \|R\mathbf{p}'_i - \mathbf{x}'_i\|^2 &= \sum_{i=1}^{N_p} \langle R\mathbf{p}'_i - \mathbf{x}'_i, R\mathbf{p}'_i - \mathbf{x}'_i \rangle \\
&= \sum_{i=1}^{N_p} \langle R\mathbf{p}'_i, R\mathbf{p}'_i \rangle - 2 \sum_{i=1}^{N_p} \langle R\mathbf{p}'_i, \mathbf{x}'_i \rangle + \sum_{i=1}^{N_p} \langle \mathbf{x}'_i, \mathbf{x}'_i \rangle \\
&= \sum_{i=1}^{N_p} RR^T \|\mathbf{p}'_i\|^2 - 2 \sum_{i=1}^{N_p} \langle R\mathbf{p}'_i, \mathbf{x}'_i \rangle + \sum_{i=1}^{N_p} \|\mathbf{x}'_i\|^2 \\
&= \sum_{i=1}^{N_p} (\|\mathbf{p}'_i\|^2 + \|\mathbf{x}'_i\|^2) - 2 \sum_{i=1}^{N_p} \langle R\mathbf{p}'_i, \mathbf{x}'_i \rangle.
\end{aligned}$$

El primer término no depende de la rotación por lo que no influirá en el cálculo. Y el segundo, que sí depende de R , hará que $\sum_{i=1}^{N_p} \|R\mathbf{p}'_i - \mathbf{x}'_i\|^2$ sea mínimo cuando tome su valor máximo al estar restando. Nos encontramos ahora en un problema de maximización. Usamos ahora la representación en cuaternios de las rotaciones y que hemos introducido en la sección 6.1.3. Sea q el cuaternion (unitario) que representa la misma rotación que la matriz R . Tenemos que maximizar:

$$\sum_{i=1}^{N_p} \langle q\mathbf{p}'_i q^*, \mathbf{x}'_i \rangle.$$

Aplicando la definición del producto de cuaternios no es difícil probar que:

$$\sum_{i=1}^{N_p} \langle q\mathbf{p}'_i q^*, \mathbf{x}'_i \rangle = \sum_{i=1}^{N_p} \langle q\mathbf{p}'_i, \mathbf{x}'_i q \rangle. \quad (6.4)$$

Visto q como un vector de \mathbb{R}^4 tenemos que $q = (q_0, q_1, q_2, q_3)^T$. También vemos los puntos \mathbf{p}'_i y \mathbf{x}'_i como cuaternios puros notando a sus componentes como $\mathbf{p}'_i = (0, p'_{i1}, p'_{i2}, p'_{i3})$ y $\mathbf{x}'_i = (0, x'_{i1}, x'_{i2}, x'_{i3})$ para $i = 1, \dots, N_p$.

Nuestro propósito ahora es escribir los sumandos de (6.4) como producto matricial. Construimos las matrices para cada $i = 1, \dots, N_p$:

$$P_i = \begin{pmatrix} 0 & -p'_{i1} & -p'_{i2} & -p'_{i3} \\ p'_{i1} & 0 & p'_{i3} & -p'_{i2} \\ p'_{i2} & -p'_{i3} & 0 & p'_{i1} \\ p'_{i3} & p'_{i2} & -p'_{i1} & 0 \end{pmatrix}, \quad X_i = \begin{pmatrix} 0 & -x'_{i1} & -x'_{i2} & -x'_{i3} \\ x'_{i1} & 0 & -x'_{i3} & x'_{i2} \\ x'_{i2} & x'_{i3} & 0 & -x'_{i1} \\ x'_{i3} & -x'_{i2} & x'_{i1} & 0 \end{pmatrix}. \quad (6.5)$$

Observamos que $q\mathbf{p}'_i$ coincide con $P_i q$ y que $\mathbf{x}'_i q$ hace lo mismo con $X_i q$, sustituimos

en (6.4):

$$\begin{aligned} \sum_{i=1}^{N_p} \langle q p'_i, x'_i q \rangle &= \sum_{i=1}^{N_p} \langle P_i q, X_i x \rangle \\ &= \sum_{i=1}^{N_p} \langle q, P_i^T X_i q \rangle \\ &= \langle q, \left(\sum_{i=1}^{N_p} P_i^T X_i \right) q \rangle. \end{aligned}$$

Podemos comprobar sin dificultad que la matriz $P_i^T X_i$ es simétrica por lo que la suma de ellas también lo es. Llameamos a su suma

$$M = \sum_{i=1}^{N_p} P_i^T X_i. \quad (6.6)$$

Como M es simétrica podemos asegurar que todos sus valores propios son reales. Sean $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4$ dichos valores propios (tenemos en cuenta las multiplicidades) y $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4$ sus correspondientes vectores propios unitarios asociados. Sabemos que vectores propios asociados a diferentes valores propios son perpendiculares entre sí. Si corresponden al mismo podemos elegirlos de tal manera para que también sean ortogonales. Por lo tanto, podemos escribir el cuaternionio q como combinación lineal de vectores propios:

$$q = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + \alpha_4 \mathbf{v}_4,$$

tenemos entonces que la matriz M expresada en la base $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\}$ de sus vectores propios es diagonal con $M = \text{diag}(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$. Así:

$$\begin{aligned} \langle q, Mq \rangle &= \langle \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + \alpha_4 \mathbf{v}_4, M(\alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + \alpha_4 \mathbf{v}_4) \rangle \\ &= \langle \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + \alpha_4 \mathbf{v}_4, \lambda_1 \alpha_1 \mathbf{v}_1 + \lambda_2 \alpha_2 \mathbf{v}_2 + \lambda_3 \alpha_3 \mathbf{v}_3 + \lambda_4 \alpha_4 \mathbf{v}_4 \rangle \\ &= \lambda_1 \alpha_1^2 + \lambda_2 \alpha_2^2 + \lambda_3 \alpha_3^2 + \lambda_4 \alpha_4^2. \end{aligned}$$

De este modo, $\langle q, Mq \rangle$ es máximo cuando $\alpha_1^2 = 1$ y $\alpha_2^2 = \alpha_3^2 = \alpha_4^2 = 0$ ya que al ser q unitario debe cumplir $\alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2 = 1$. Así, el cuaternionio que maximiza (6.4) es el vector propio asociado al máximo valor propio de M .

6.3. Algoritmo ICP

Una vez explicadas las herramientas que intervienen en el proceso de minimización de la función (6.2), explicamos el algoritmo tal y como fue propuesto por Paul J.Besl and Neil D.MacKay. Destacamos una sutil diferencia entre la construcción original y la que nosotros hemos presentado. Besl and MacKay identifican la matriz M que hemos definido en (6.6) como:

$$M = Q(\Sigma_{px}) = \begin{pmatrix} \text{tr}(\Sigma_{px}) & \Delta^t \\ \Delta & \Sigma_{px} + \Sigma_{px}^t + \text{tr}(\Sigma_{px})I_3 \end{pmatrix}, \quad (6.7)$$

donde

$$\Sigma_{px} = \frac{1}{N_p} \sum_{i=1}^{N_p} [(p_i - \bar{p})(x_i - \bar{x})^t],$$

$$A_{ij} = (\Sigma_{px} - \Sigma_{px}^t)_{ij} \quad i, j \in \{0, 1, 2\},$$

y

$$\Delta = \begin{pmatrix} A_{12} & A_{20} & A_{01} \end{pmatrix}.$$

No es difícil comprobar que ambas definiciones son equivalentes haciendo la construcción elemento a elemento. Destacar que se han presentado estas dos definiciones de la misma matriz ya que la dada por (6.6) es más fácil de comprender en el proceso de minimización. Por su parte, (6.7) será la que se implementará en el código del algoritmo al ser la propuesta original.

Así, con todo los mecanismos a nuestra disposición, la solución propuesta para alinear ambas nubes de puntos es la siguiente:

1. Sea el conjunto de puntos P a alinear con el modelo X . Notar que no tienen porqué tener el mismo tamaño.
2. Inicializaremos el conjunto $P_0 = P$. Para la iteración $k \geq 0$, repetir a), b) y c) hasta que haya convergencia con una tolerancia τ .
 - a) Calcular el conjunto de Y_k de los puntos más cercanos de P_k respecto a X . Notamos como e_k a la distancia media entre Y_k y P_k obtenida mediante (6.2).
 - b) Aplicar el método de optimización de la distancia media (sección 6.2) para obtener los cuaternios qT_k y qR_k .
 - c) Obtener el conjunto $P_{k+1} = R(qR_k)P_0 + qT_k$. Calcular d_k que nota la distancia media entre Y_k y P_{k+1} obtenida mediante (6.2).
 - d) Terminar cuando $d_{k-1} - d_k < \tau$.

En el pseudocódigo hemos notado como $R(qR_k)$ a la rotación asociada al cuaternion qR_k de la iteración k -ésima. Notar que cada $y_{i,k}$ es el punto más cercano asociado a $p_{i,k}$ para $i = 1, \dots, N_p$. De este modo, queda clara la nota que apuntamos al inicio de la sección y es que podemos suponer que los conjuntos tienen el mismo número de puntos ya que el proceso no se realiza con el modelo X sino con el conjunto de puntos más cercanos.

Exponemos ahora el teorema que nos asegura la convergencia del método sugerido, que en este caso es convergencia local. Esto nos impondrá una serie de restricciones que explicaremos más adelante.

Teorema 6.2. *El algoritmo ICP siempre converge de manera monótona a un mínimo local respecto de la función objetivo distancia media.*

Demostración. Tenemos el conjunto P_0 que queremos ajustar al modelo X . Sea $P_k = R(qR_{k-1})P_0 + qT_{k-1}$ el conjunto de puntos transformados al principio de la iteración k y sea Y_k el conjunto de puntos más cercanos a cada punto con $y_{i,k}$

el punto más cercano asociado a $p_{i,k}$ para $i = 1, \dots, N_p$ que hemos calculado. Definimos:

$$e_k = \frac{1}{N_p} \sum_{i=1}^{N_p} \|y_{i,k} - p_{i,k}\|.$$

Aplicamos el proceso explicado en la sección 6.2 y obtenemos qT_k y qR_k . Calculamos:

$$d_k = \frac{1}{N_p} \sum_{i=1}^{N_p} \|y_{i,k} - R(qR_k)p_{i,k} - qT_k\|.$$

Afirmamos que $d_k \leq e_k$ para la iteración k -ésima. De lo contrario, si $d_k > e_k$ tendríamos que la distancia media del conjunto inicial es menor que la distancia media tras aplicar el proceso descrito de optimización lo cual es imposible por la propia construcción del mismo.

En la siguiente iteración, calculamos $P_{k+1} = R(qR_k)P_k + qT_k$. Obtenemos ahora el Y_{k+1} y e_{k+1} igual que en la anterior iteración. Es evidente que $e_{k+1} \leq d_k$ por la propia definición de Y_{k+1} . Notar que para toda iteración k se cumple $e_k, d_k \geq 0$ al ser la suma de números no negativos por lo que hemos conseguido la siguiente relación:

$$0 \leq d_{k+1} \leq e_{k+1} \leq d_k \leq e_k$$

Finalmente como la sucesión $\{d_k\}_{k \geq 1}$ es decreciente y acotada podemos afirmar que es convergente y como consecuencia que el método converge de manera monótona a un mínimo local. ■

Una vez demostrada la convergencia del método, es necesario recalcar una serie de observaciones acerca del mismo.

Observación 6.5. *Al converger a un mínimo local es necesario una fase de prealineado*

La demostración del método asegura que el método es convergente pero no nos aporta el elemento al que converge. Por ello, al aplicar el método podemos llegar a una situación donde el procedimiento converja a una mejora en la transformación pero no sea la que nosotros deseamos. Necesitamos entonces la fase de *prealineado*. Este proceso se puede llevar a cabo de diferentes maneras. La más habitual es dar manualmente una serie de puntos clave que se correspondan en los dos conjuntos y aplicar una transformación de acuerdo a los mismos. Tenemos así una primera aproximación de ambos conjuntos de puntos para asegurar que la convergencia se produce a la situación que deseamos. En nuestro caso, dicho procedimiento se ha implementado del siguiente modo:

1. Escoger de cada conjunto tres puntos que se encuentren en ambos y sean fácilmente identificables al pertenecer a una esquina, tener el mismo color, etc. Estos puntos deben de estar ordenados de la misma forma en ambos casos, es decir, el primer punto seleccionado de un conjunto debe “coincidir” con el primero del otro. Igual con los dos restantes. Este proceso se recoge en la figura 6.2.

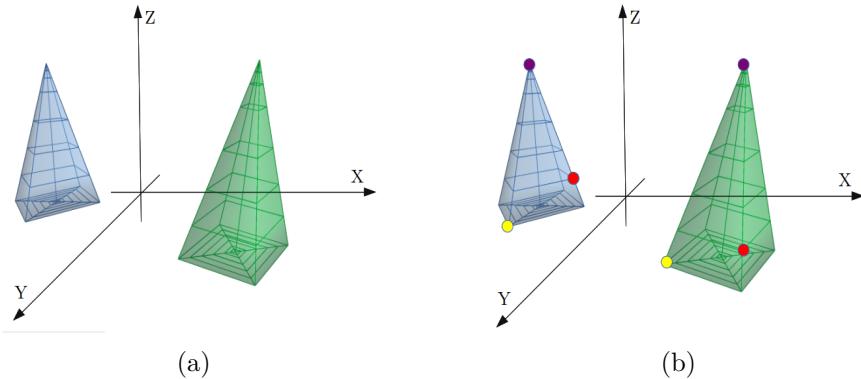


Figura 6.2: Selección de puntos en ambos conjuntos.

2. Llevar los primeros puntos seleccionados al origen de coordenadas.
3. Llevar los segundos a la parte positiva del eje X manteniendo fijos los anteriores.
4. Intentar aproximar lo máximo posible los otros puntos mediante una rotación alrededor del eje X .

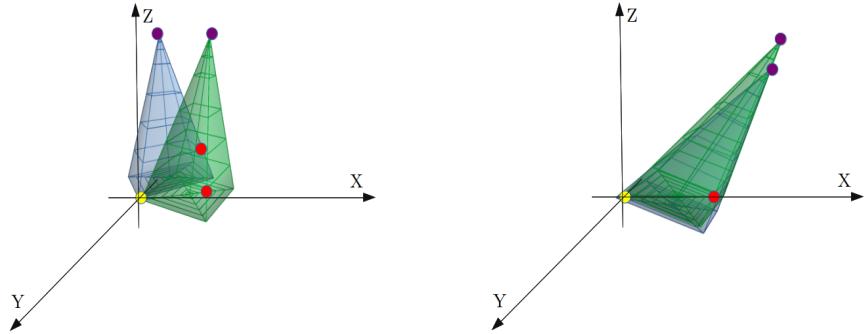
La figura 6.3 muestra el proceso a seguir en estos últimos pasos. En secciones posteriores veremos un ejemplo práctico de este procedimiento.

Observación 6.6. *El cálculo de los puntos más cercanos entre las nubes de puntos es un proceso muy costoso en tiempo.*

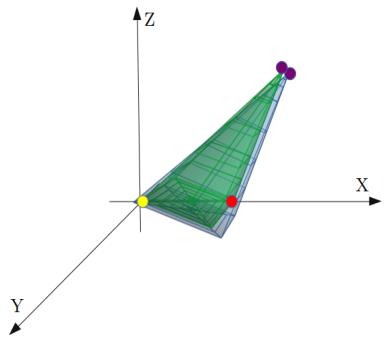
Este paso del algoritmo es el que más tiempo ocupa de todo el proceso al tener una complejidad $O(N_p^2)$ en el peor de los casos. También tenemos que tener en cuenta el tamaño de los conjuntos de los puntos. La finalidad de este proceso es tener un modelo digital detallado y fiel a la realidad de un objeto real. Ello conlleva que las tomas se realicen con cierta calidad lo que hace que, incluso en modelos relativamente pequeños, cada una ronde los 2 millones de puntos. Si es un modelo más complejo podríamos llegar a los 20 millones. En estas situaciones, aplicar el método tradicional podría durar horas en completarse.

En esta situación, nos planteamos la posibilidad de una variante para no tener que comprobar todos y cada uno de los puntos en el proceso. En una primera aproximación, podríamos pensar en coger una subconjunto aleatorio de puntos de cada toma y aplicar el proceso descrito. Sin embargo, enseguida comprobamos que esta solución no es válida. Esto se debe a la convergencia local del método lo que nos conduce a las situaciones indeseadas que queremos evitar con el prealineado. Por ello, nos tenemos que plantear mecanismos más complejos que aseguren que se tengan en cuenta las tomas completas.

Un primer método sería usar índices espaciales como los *octrees*. Los *octrees* son estructuras jerárquicas que dividen el espacio en cubos denominados *voxels*. Cada *voxel* puede contener un máximo número de puntos por lo que si se supera



(a) Paso 2: llevar los primeros puntos al (b) Paso 3: llevar los segundos puntos al
origen.



(c) Paso 4: aproximar lo máximo posible
el punto restante.

Figura 6.3: Mecanismo de prealineado a partir de tres puntos.

dicho número el *voxel* se dividiría en otros *voxels* más pequeños. También podríamos imponer un profundidad máxima del árbol. En esta situación, solo deberíamos comprobar la distancia con un punto de cada *voxel* y una vez se tiene, filtrar los puntos del modelo por esa distancia.

Otro mecanismo sería identificar alguna característica que nos permita clasificar los puntos. Como las tomas corresponden al mismo modelo, podemos suponer que la característica que buscamos se mantienen en cierta medida en ambos casos y solo tenemos que buscar el punto más cercano con aquellos que tienen características similares. Esta será la opción que nosotros usaremos para proponer alternativas basadas en este procedimiento y que veremos en la sección 7.1 .

6.4. Ejemplos prácticos

En esta sección procedemos a probar el método implementado. Empezamos con un ejemplo parecido al artículo original. Los conjuntos de puntos para la prueba

son:

x_1	x_2	x_3
0.4389	-0.0588	1.0699
0.4202	0.2052	1.1252
0.4201	0.2539	1.1325
0.4495	0.0469	1.1260
0.4412	0.1796	1.1515
0.4826	-0.0137	1.1359
0.4628	0.0703	1.1458
0.4700	0.1852	1.1765

Tabla 6.1: Conjunto 1.

y_1	y_2	y_3
0.7278	0.0712	1.4610
0.7019	0.2480	1.4867
0.7621	0.1828	1.4720
0.7271	0.1769	1.4809
0.7067	0.0462	1.495
0.6438	-0.0540	1.4359
0.7247	-0.0116	1.4385
0.6982	0.1981	1.4832
0.7700	0.2500	1.5000
0.8000	-0.100	1.400
0.8300	0.3000	1.4500

Tabla 6.2: Conjunto 2.

En la imagen 6.4 se muestran los puntos antes y después de aplicar una iteración de ICP. Notar que los del primer conjunto se han dibujado de color rojo y los del segundo de color verde. En esta ocasión no ha sido necesaria la etapa de prealineado al no haber una situación deseada tras la transformación.

Según los datos calculados, la distancia media a pasado de 0,162697 a 0,00372463. El tiempo que ha tardado en realizar el proceso es de 0,00226548 segundos. La translación y rotación expresada como cuaternion vienen dadas por

$$q_T = (0,166271, 0,165485, 0,349303).$$

$$q_R = (0,0234481, 0,0238614, -0,154144, 0,987482).$$

El segundo ejemplo que vamos a probar consiste en un caso real, concretamente dos tomas de una escultura de un pie perteneciente a la Facultad de Bellas Artes de la Universidad de Granada tomadas con un escáner láser Faro Focus 130. Queremos alinear dos tomas obtenidas desde diferentes ángulos:

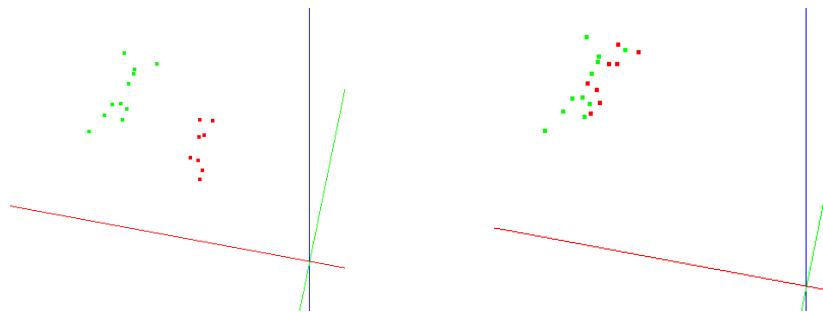
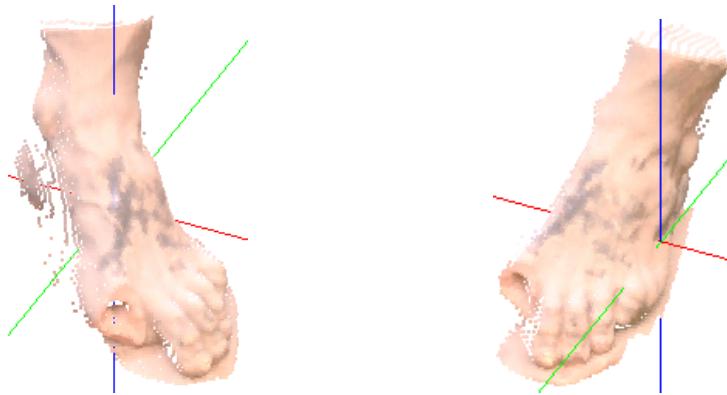


Figura 6.4: Conjuntos de puntos antes y después de una iteración de ICP.



Conjunto (1) con 7 042 puntos.

Conjunto (2) con 8 334 puntos.

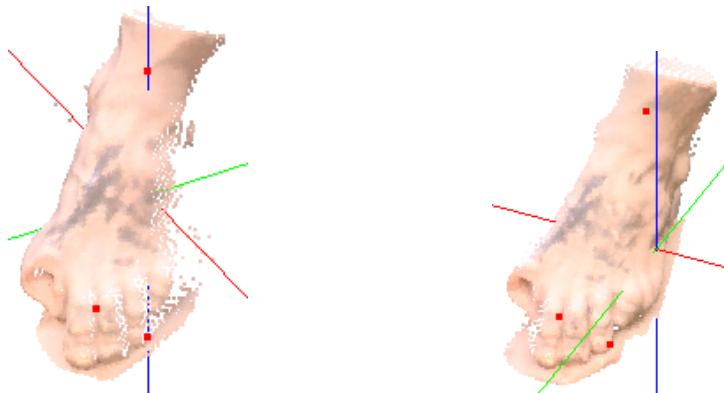
Figura 6.5: Tomas a alinear.

Tras la etapa de prealineado (los puntos seleccionados aparecen en la figura 6.6) se ha obtenido el resultado que muestran la figura 6.7.

A continuación, comenzamos usamos el algoritmo ICP. La tabla 6.3 recoge los datos tras cada iteración. Destacar que con distancia nos referimos a la distancia media entre los puntos y los más cercanos, es decir, lo que hemos notado como e_k y d_k en la sección 6.3. Por su parte, la figura 6.8, muestra el resultado final tras las cinco iteraciones llevadas a cabo.

Iteración	Dist. antes (m^2)	Dist. después (m^2)	Segundos
1	0.000449468	0.000328695	62.8672
2	0.00027947	0.000251878	62.7888
3	0.000229901	0.000209142	62.8636
4	0.000195308	0.000188837	63.059
5	0.000186373	0.000185154	62.8259

Tabla 6.3: Resultados ajuste mediante ICP.



Puntos clave del primer conjunto. Puntos clave del segundo conjunto.

Figura 6.6: Etapa de prealineado.

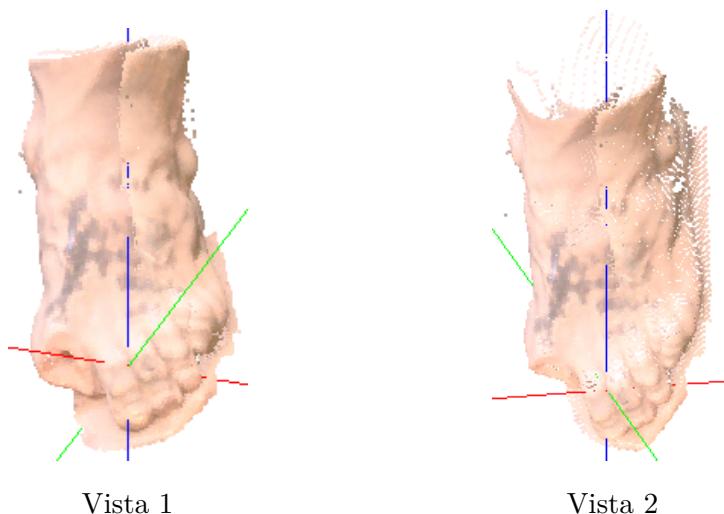


Figura 6.7: Vistas tras prealinear.

En conclusión, vemos cómo el programa propuesto ha sido capaz de realizar un buen ajuste y por ello hemos conseguido un modelo más completo al original uniendo ambas tomas. Destacamos nuevamente la importancia que tiene en el proceso el prealineado. No solo na necesidad de hacerlo sino también la manera en la que sea lleva a cabo este proceso. Es posible que incluso incluyendo esta etapa el resultado no sea el deseado. Este comportamiento se debe como ya hemos indicado a la convergencia local del método ICP. Las dos nubes de puntos tienen que estar los no solo lo suficientemente cerca sino también colocadas de tal manera que el algoritmo propuesto solo deba realizar un pequeño ajuste.

Por otro lado, también vemos que el tiempo empleado es elevado tal y como se indicó en la observación 6.6: cada iteración ha durado algo más de un minuto.

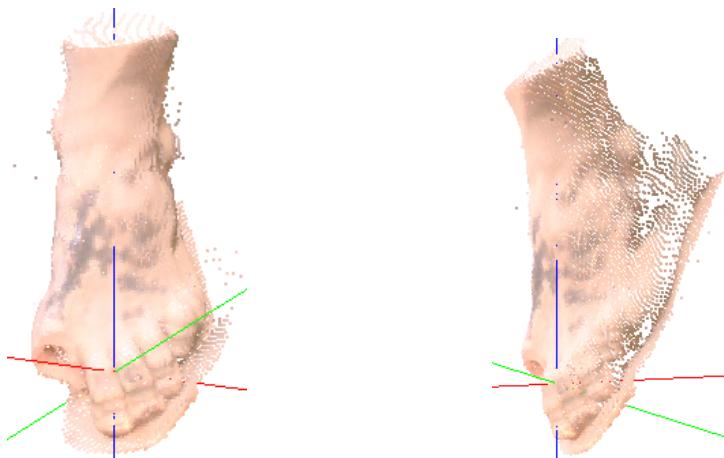


Figura 6.8: Vistas tras cinco iteraciones del ICP.

Hay que tener en cuenta que las nubes de puntos tomadas como modelo tienen un tamaño muy pequeño. En situaciones reales las tomas pueden tener millones de puntos lo que conllevaría mucho tiempo a la hora de aplicar el algoritmo e incluso sin la certeza de si el resultado es el que deseamos. En la siguiente sección planteamos unos métodos para acelerar este proceso.

Cambios en el método: uso de normales

En las sección anterior nos encontramos con el siguiente problema: el método ICP funciona correctamente pero es demasiado lento ya que tiene un orden cuadrático y manejamos gran cantidad de información. Nuestro objetivo es entonces reducir el conjunto de puntos con los que trabajamos sin perder información. Este proceso no puede ser aleatorio y debe ser lo más significativo posible ya que, de lo contrario, podríamos obtener un resultado no deseado. Así usaremos en estos procedimientos los denominados *descriptores*. Un descriptor lo podríamos definir como una medida de un punto en base a cierta característica que escojamos. De este modo obtenemos los llamados *puntos significativos*, *puntos clave* o *key points* de cada una de las tomas. Como en nuestro marco de trabajo los modelos que tomamos como base son rígidos, es decir, no hay movimiento, cambios de escala, etc. podemos suponer que los puntos clave detectados en una toma deben de ser los mismos que en la otra. Por lo tanto, en una primera aproximación, si realizamos el proceso para buscar la mejor transformación solo con los puntos clave deberíamos conseguir nuestro objetivo.

Como observación, aclarar que del mismo modo que no existe un método que funcione correctamente en el alineado de cualquier modelo, no existe un descriptor que funcione de manera adecuada con todos los modelos. Por ejemplo, podríamos detectar puntos clave en base al color. Si nos encontramos ante una escultura con un ropaje colorido y detallado es un descriptor que podríamos considerar válido. Sin embargo, si nuestro modelo presenta un color plano o una variación poco significativa del mismo no conseguiremos obtener puntos significativos. También debemos de tener en cuenta el tiempo necesario para la detección de puntos clave. El proceso debe ser lo suficientemente rápido para que reduzcamos el tiempo de alineación de las tomas y lo suficientemente bueno para que el conjunto de puntos calculado sea pequeño y descriptivo dentro de toda la nube.

7.1. Variación de la normal

El método para la obtención de puntos consistirá en la detección de zonas donde se produzca un cambio brusco de la normal calculando de este modo esquinas, hendiduras, etc. Este procedimiento no es válido en superficies suaves ya que la

variación de la normal en un entorno sería parecida en todo el modelo. Situación de este tipo nos la podemos encontrar, por ejemplo, en esferas. Una vez explicada la idea principal procedemos a explicar el proceso llevado a cabo.

En primer lugar, debemos calcular las normales de cada punto de la nube. Generalmente, es un proceso costoso aunque en nuestra situación tenemos una gran ventaja. La información aportada por escáner con el que tomamos cada una de la muestras nos aporta, en cierta medida, una malla implícita ya que que puede entregar los datos de los puntos registrados siguiendo una cuadrícula en un archivo con formato PTX, que nos aporta el resultado por columnas por lo que podemos reconstruir la “matriz” espacial calculada previamente. Así con esta información para cada punto calculamos la normal mediante el producto vectorial de los triángulos adyacentes y luego promediamos obteniendo la de cada punto. En la figura 7.1, vemos a la izquierda la malla implícita que obtenemos con el formato que estamos trabajando. A la derecha aparece un ejemplo de cómo se han calculado las normales. En amarillo aparecen los vectores que se calculan (siempre y cuando existan dichos puntos) y en rojo los productos vectoriales necesarios. Destacamos que es importante que todas las normales tengan un convenio general de apuntar hacia dentro o hacia fuera del modelo. El nuestro ha sido de que apunten hacia afuera, lo que se ha tenido en cuenta durante la obtención de los productos vectoriales.

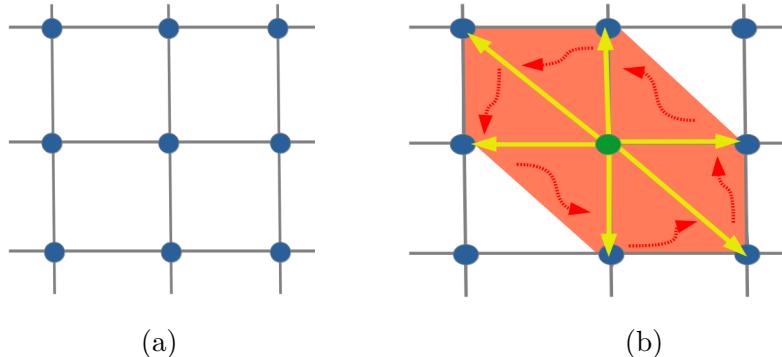


Figura 7.1: (a) Malla implícita, (b) Cálculo de la normal.

La idea del algoritmo propuesto se basa en que en zonas relativamente planas las normales son muy parecidas por lo que la suma los productos escalares de la normal en un punto con aquellos contenidos en un entorno suyo es prácticamente 0. Razonemos lo que pasa en los lugares en los que hay esquinas. Tomamos un punto y sus vecinos, que en general son un total de 8. Suponemos una situación ideal en la que el punto que hemos escogido está en un borde. Si es así, consideraremos que dos de los puntos adyacentes también están en dicho borde. Así, tenemos que de sus vecinos adyacentes, dos tienen una normal parecida y los seis restantes perpendicular. Si calculamos el producto escalar medio en esta situación ideal obtenemos que es de 0,25. Por ello, el algoritmo propuesto tiene en cuenta este valor y solo se queda con los puntos cuyo descriptor de normales esté en un rango próximo al valor comentado, que pertenecería a la situación ideal. En nuestros experimentos hemos usado como criterio que el valor del producto escalar medio esté entre 0,15

y 0,3.

A continuación mostramos un ejemplo de los resultados obtenidos. Para ello, hemos usado diferentes niveles de precisión. Esto se ha conseguido simplificando la malla implícita que tenemos del modelo e indicando el número de filas y columnas que queremos conservar. Por ejemplo, si tenemos un nivel de precisión de 5, toma como puntos válidos aquellos que están cada 5 filas y 5 columnas. De este modo, de cada 25 puntos nos quedamos con uno. Si el nivel de precisión es 8, nos quedamos con un punto de 64. En la figura 7.2 se muestran varios ejemplos de cómo varía la densidad de la malla en función del nivel de simplificado.

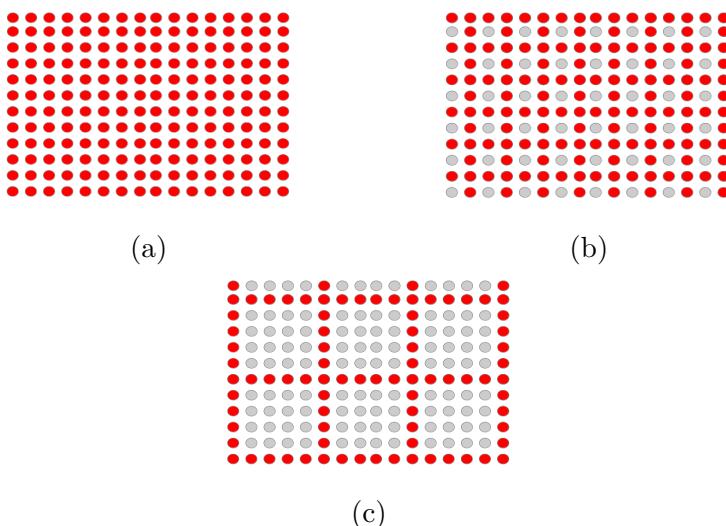


Figura 7.2: Ejemplos de simplificado de la malla: (a) sin simplificado, (b) valor 2 y (c) valor 5. En rojo aparecen marcados los puntos con los que nos quedamos en cada caso.

Los resultados obtenidos tras aplicar el algoritmo de detección de puntos claves se muestran en la tabla 7.1 y figura 7.3. Vemos que uno de los problemas que tiene este método es que muy sensible al ruido. Cuanto más preciso sea el modelo base menos adecuado es el descriptor propuesto. Por lo tanto, si queremos que los puntos obtenidos tras el proceso sean significativos tenemos que calcularlos a partir del modelo simplificado. Esto no debe de ser una problema ya que esto solo lo usaremos en las primeras iteraciones del método ICP ya que para obtener la solución deseada debemos tener en cuenta posteriormente el modelo entero debido a la sensibilidad del procedimiento.

7.2. Pruebas

7.2.1. De puntos clave a todo el modelo

La primera prueba que se va a realizar es aplicar el procedimiento a los puntos clave pero tomados de parte de una toma a una otra toma completa que tomamos

Nivel de simplificado	Puntos totales	Tiempo cálculo normales (seg.)	Tiempo cálculo ptos. clave (seg.)	Puntos clave
1	2 275 886	25.1373	13.0592	592 611
2	568 830	6.39203	3.26905	192 031
3	252 881	2.82	1.47162	46 610
4	142 186	1.72909	0.837061	11 866
5	90 983	1.00018	0.549648	4490
8	35 519	0.394472	0.215125	1 885

Tabla 7.1: Resultados del cálculo de puntos clave. En la primera columna aparece el nivel de simplificado utilizado, en la segunda el número total de puntos de la malla (simplificada), en la tercera y cuarta se incluye el tiempo en segundos para el cálculo de las normales y el cálculo de los valores del descriptor respectivamente. En la última columna se aporta el número de puntos clave obtenidos en cada caso.

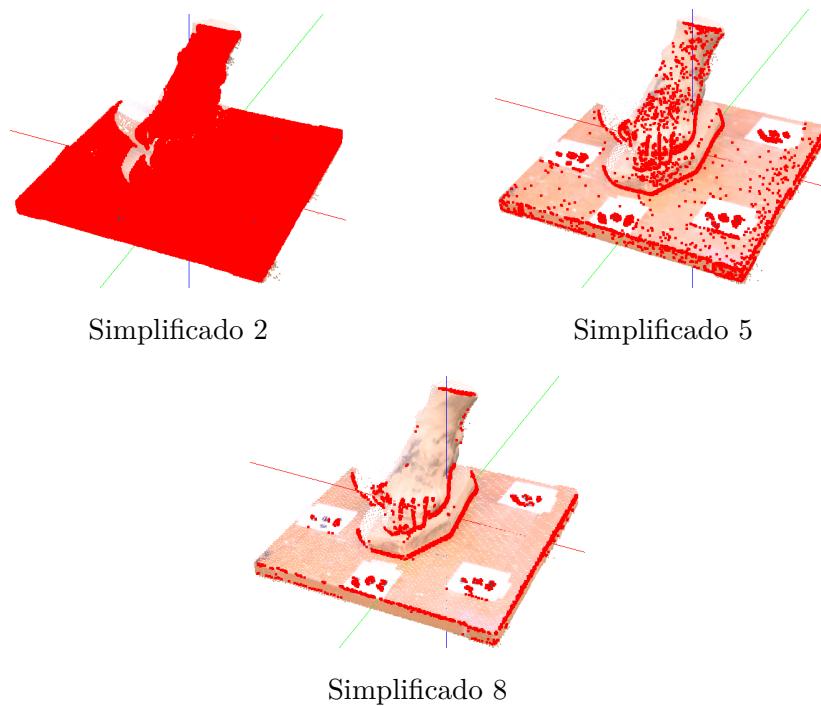
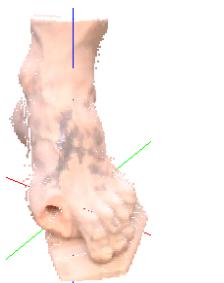


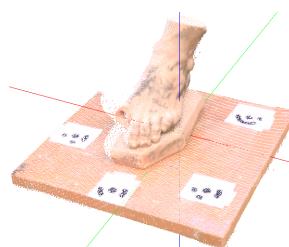
Figura 7.3: Descriptor normales según el nivel de simplificado.

como modelo. Lo explicamos con nuestro ejemplo base del pie. Si probamos con las mismas tomas de la sección 6.4 en las que solo aparece el pie los puntos que nos aportará el descriptor mediante normales es posible que no tengan una correspondencia clara con la otra toma, sobre todo, los que se encuentran en el borde. Por eso, debemos tomar como modelo la toma completa para asegurar en todo lo que posi-

ble que haya correspondencia entre los puntos clave aportados. Del mismo modo, también habría problemas si tomamos las dos tomas completas ya que nuevamente es probable que algunos puntos no estén en el modelo, y al intentar ajustarlos, nos aporte una solución errónea. Destacamos que aplicar el procedimiento propuesto a una parte de las tomas no debe de ser restrictivo porque nos interesa conocer la transformación. Podríamos almacenar los datos de las transformaciones que se van realizando y posteriormente aplicarlo a la toma completa. Otra solución sería, por ejemplo, imponer una distancia máxima entre dos puntos para asegurarse de que realmente se corresponden. Esta solución será la que usaremos en la siguiente sección.



Conjunto (1) con 7 527 puntos.



Conjunto (2) con 79 921 puntos.

Figura 7.4: Tomas a alinear.

El número de puntos característicos detectados en la toma de la izquierda mediante el descriptor de las normales en el intervalo (0.15, 0.3) es de 506. Aplicamos el procedimiento y los resultados obtenidos se muestran en la figura 7.5 y en la tabla 7.2.

Iteración	Dist. antes (m^2)	Dist. después (m^2)	Segundos
1	0.000545768	0.00028771	20.0778
2	0.000260278	0.000240228	19.7434
3	0.000231921	0.00022784	19.4636
4	0.000224222	0.00022315	19.4807

Tabla 7.2: Resultados ajuste mediante ICP.

Vemos que el procedimiento funciona correctamente y además que el tiempo empleado se reduce considerablemente respecto a los tiempos que habríamos necesitado considerando los conjuntos completos. Para hacernos una idea del tiempo que habría tardado basta tomar el ejemplo del pie de la sección 6.4. En ese caso, los conjuntos tenían unos 7 000 y 8 000 puntos cada uno y tardaba poco más de un minuto. Ahora unos de ellos tiene casi 80 000, por lo que teniendo en cuenta el orden cuadrático del método, incrementaría bastante el tiempo necesario para completar cada iteración. Destacar que al igual que el procedimiento original, es posible que el resultado obtenido no sea el deseado y tenga que aplicarse desde el principio.

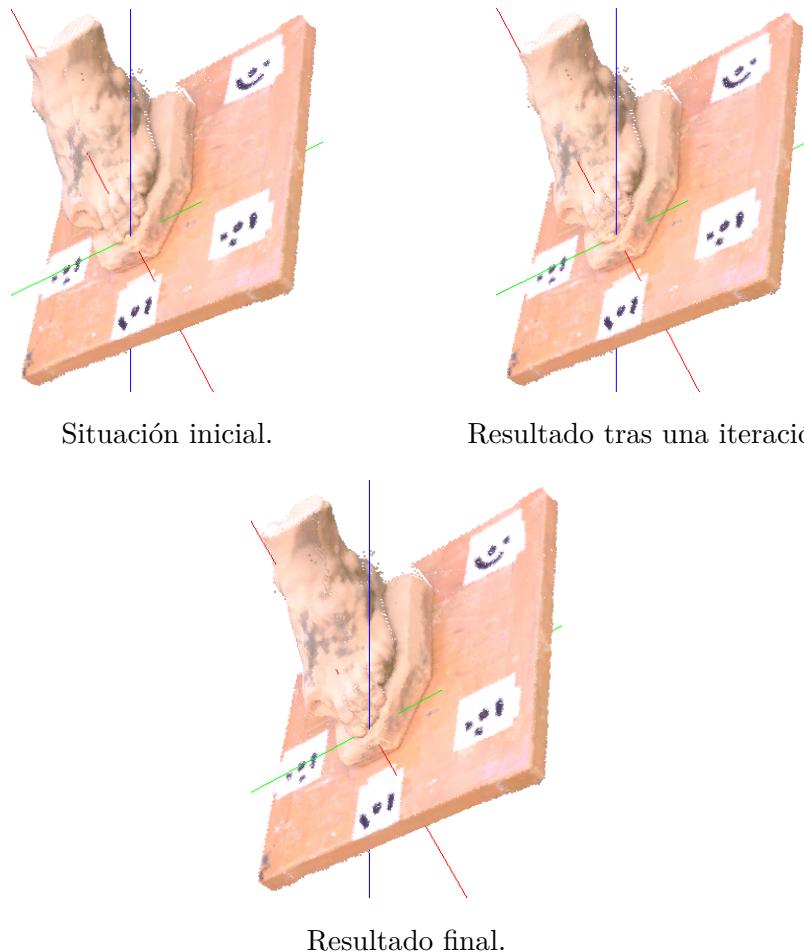


Figura 7.5: Algoritmo ICP usando un conjunto de puntos clave.

7.2.2. De puntos clave a puntos clave

Nuestro propósito ahora es intentar reducir aún más el conjunto de puntos con los que realizar el proceso ICP. Para ello, vamos a probar qué pasa si aplicamos el algoritmo solamente a los conjuntos de puntos clave calculados. Este paso hay que hacerlo con cuidado ya que, incluso en mayor medida que tomando en una muestra la toma entera, nos arriesgamos a que muchos puntos no tengan una correspondencia. Así, para el cálculo del punto más cercano se ha tenido en cuenta:

1. Solo se comprueban puntos clave con un valor de descriptor parecido. Así, se ha fijado una diferencia máxima permitida y solo se comprueban los puntos que tengan un valor dentro de ese intervalo. Este se debe a que al ser movimientos rígidos no hay ningún tipo de deformación por lo que los planos serán los “mismos” mismos entre una toma y otra. Sin embargo, ponemos un intervalo debido a los errores del escáner y a las pérdidas de precisión debido al simplificado del modelo para la obtención de puntos clave.

2. Se ha acotado la distancia máxima permitida para un punto. Así, una vez calculada la distancia mínima de un punto al otro conjunto comprobamos que esa distancia no supere un umbral que hemos prefijado. Si lo supera significa que el punto más cercano está demasiado lejos y por ello no se corresponden realmente. De este modo, nos aseguramos que zonas que no se solapen se intenten ajustar y acabemos teniendo una solución errónea.

En la figura 7.6 mostramos un ejemplo del resultado obtenido imponiendo que los valores de los descriptores entre los puntos clavén difieran en 0,01 y que la distancia (en metros al cuadrado) entre ellos no sea mayor a 0,1.

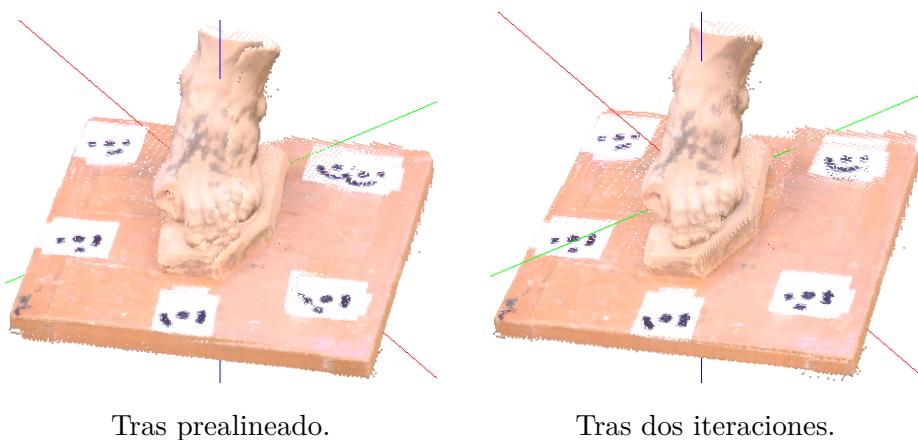


Figura 7.6: Proceso ICP solo con puntos clave.

Se han realizado dos iteraciones. En ambas ha tardado aproximadamente 0,7 segundos y la distancia media (en metros al cuadrado) ha variado de 0,000782562 a 0,000722275. Los puntos clave calculados en cada caso son en la primera toma un total de 1142 y en la segunda 2273. Observamos cómo tras las dos iteraciones se han conseguido ajustar mejor las dos tomas. Nuevamente, hacemos hincapié en que el algoritmo ha tardado menos de lo que habría necesitado en caso de haber tomado las nubes de puntos completas gracias a la reducción del número de elementos de ambos conjuntos.

Una segunda aproximación es contar el número de productos escalares que dan “cero” para saber el número de planos perpendiculares que tiene cada punto. De este modo, solo comparamos con puntos cuyo número de ceros es el mismo. Se ha tomado perpendicularidad cuando el producto escalar sea menor que 0,01 debido a errores en la toma de muestras y simplificado de la malla. Notar que seguimos teniendo en cuenta la cota de la distancia máxima. Esto puede hacer que en un paso del método la distancia pueda ser mayor que el anterior al existir un mayor o menor número de puntos que se han tomado como referencia. Esto no debería ser ningún problema *a priori*. En el ejemplo que mostramos ese número no ha variado entre iteraciones.

En la figura 7.7, vemos que nuevamente se ha obtenido un proceso satisfactorio en la zona de los dedos del pie aunque ha tardado más tiempo en cada iteración que en el caso anterior.

Iteración	Dist. antes (m^2)	Dist. después (m^2)	Segundos
1	0.00124479	0.00121367	2.33132
2	0.00119796	0.0011896	2.32062
3	0.00118431	0.00118055	2.30985
4	0.00117839	0.00117688	2.32283

Tabla 7.3: Resultados ajuste mediante ICP.

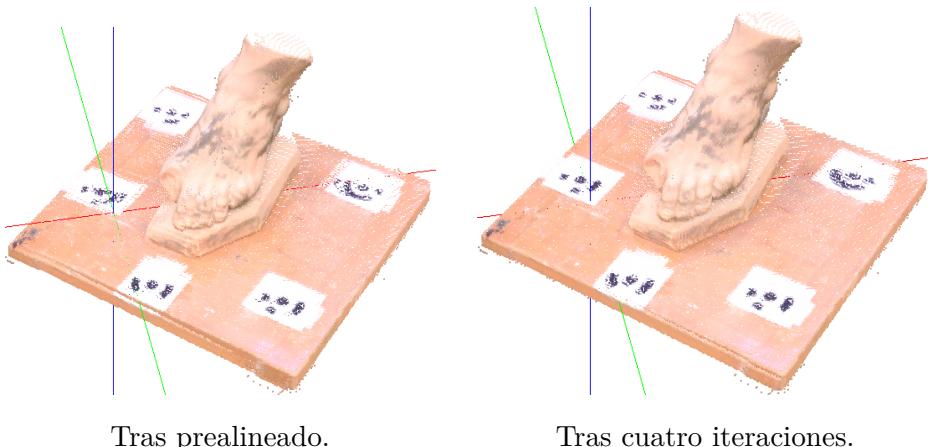


Figura 7.7: Proceso ICP solo con puntos clave agrupándolos por el número de ceros de sus productos escalares.

7.3. Torre de las gallinas

En secciones anteriores, hemos realizado diversas pruebas con un modelo de un pie sobre un tablero. En esta probaremos los métodos ya implementados con otro modelo que representa una estancia dentro de una torre. Más concretamente, se trata de una parte de la torre de las gallinas, situada en la Alhambra escaneada usando un escáner láser Faro Focus 130. En las figuras 7.8 y 7.9 se muestran las dos tomas que se quieren alinear.

Destacamos la gran cantidad de puntos que tiene cada una de las nubes: la primera de ellas consta de 10 557 654 puntos y la segunda de 10 801 863 puntos. Si aplicáramos el procedimiento ICP original necesitaríamos bastantes horas para realizar el proceso. Por ello, lo que vamos a hacer es calcular los puntos clave de las tomas y luego aplicamos el algoritmo de alineado a los puntos obtenidos.

En primer lugar, vamos a comprobar cómo se comparten las tomas para la obtención de puntos clave sin simplificado y con simplificado tal y como hemos hecho en la sección 7.1. También tendremos en cuenta el número de puntos que se detectan en cada una de ellas. El nivel de simplificado que se ha escogido ha sido 5 ya que se ha considerado que puede ser un buen valor para tener la suficiente densidad en la malla y a su vez tener un número de puntos más tratable. Veamos cómo varían los datos de tiempo y tamaño mostrados en la tabla 7.4. Notamos que se han usado las mismas cotas para el cálculo de los puntos clave que anteriormente.

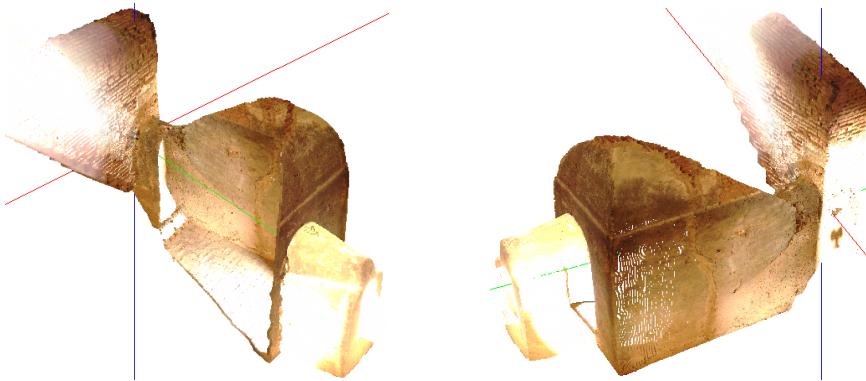


Figura 7.8: Nube de puntos de la torre de las gallinas usada como toma 1.

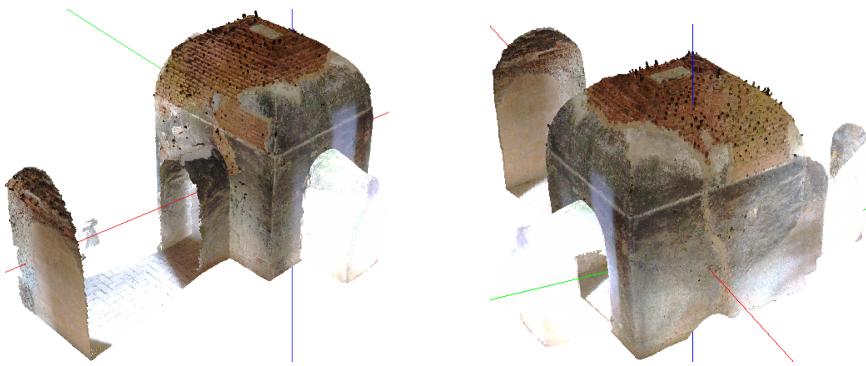


Figura 7.9: Nube de puntos de la torre de las gallinas usada como toma 2.

	Original	Simp. 5
Calc. normales (seg.)	115.886	4.62641
Ptos. clave (seg.)	62.9924	2.41946
Ptos. clave (num.)	862 225	16 412

Tabla 7.4: Comparación resultados toma 1 original y simplificada (nivel de precisión 5). En la primera columna aparecen los valores que se han medido: tiempo de cálculo en segundos, tiempo en el cálculo de puntos clave y número de puntos clave detectados. En las columnas segunda y tercera aparecen los resultados obtenidos para la toma original y para la simplificada respectivamente.

Las imágenes con los puntos clave obtenidos se muestran en 7.10 y 7.11.

Vemos que a pesar de que en la toma simplificada la densidad de puntos es menor, el algoritmo nuevamente ha sido capaz de identificar mejor las aristas que en la original. Además, se observa que los tiempos empleados para el cálculo de las normales y obtención de puntos clave es considerablemente mejor. Finalmente, en la toma original se han obtenido un total de 862 225 puntos lo que sigue siendo un valor demasiado alto para obtener resultados con el algoritmo ICP en un tiempo

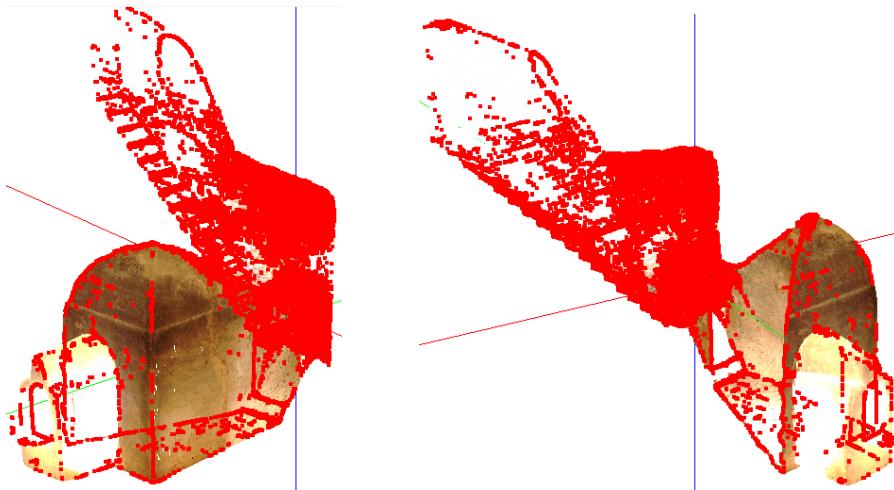


Figura 7.10: Puntos clave detectados toma 1 de la torre de las gallinas.

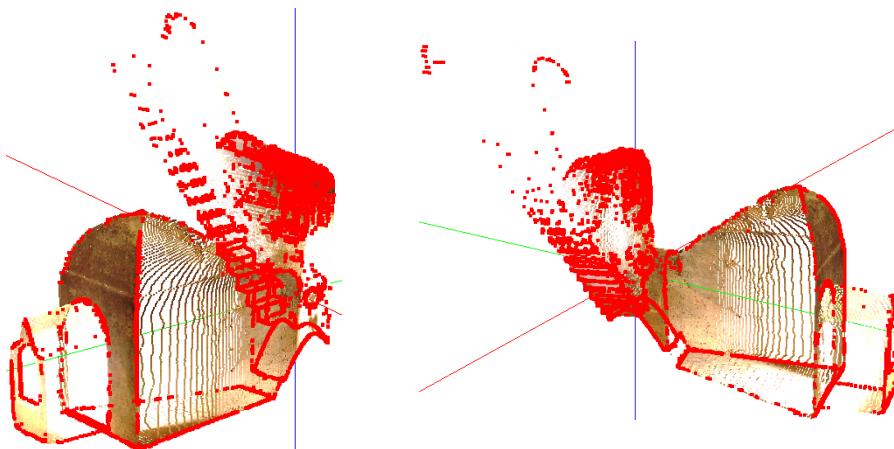


Figura 7.11: Puntos clave detectados toma 1 simplificada (nivel 5) de la torre de las gallinas.

razonable. Por otro lado, con la versión simplificada hemos obtenido 16 412. Mostramos también los puntos clave de la otra nube de puntos en la figura 7.12. Se observa que en la zona superior de la cúpula se han detectado una gran cantidad de los mismos al estar hecha de ladrillo visto. Ahora realizaremos las pruebas del algoritmo comparando los valores del descriptor para la obtención de los puntos clave, ya que, en el ejemplo del pie anterior hemos observado que tardaba menos en la ejecución de cada iteración que viendo el número de ceros de los productos escalares.

En la figura 7.13 aparecen las tomas tras el prealineado y en la 7.14 mostramos los pasos del proceso aplicando el algoritmo entre puntos clave.

Vemos cómo el método ha funcionado correctamente a la hora de hacer corres-

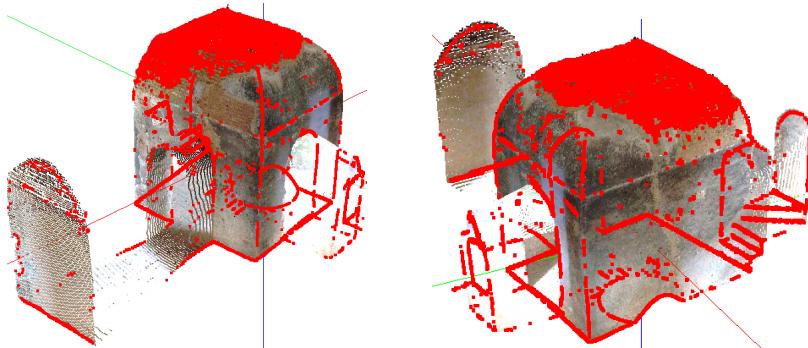


Figura 7.12: Puntos clave detectados toma 2 simplificada de la torre de las gallinas. Se han detectado un total de 33 187 puntos.

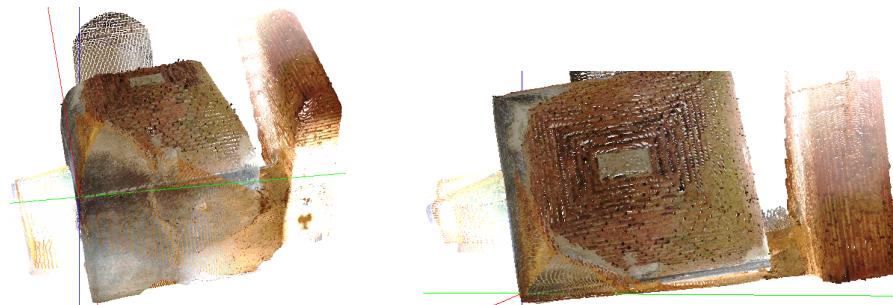


Figura 7.13: Vista general y detalle del prealineado de las tomas de la torre de las gallinas.

It.	Dist. antes (m^2)	Dist. después (m^2)	Segundos	Num. puntos
1	0.0424663	0.0413611	40.4705	1 997
2	0.0344874	0.0322639	41.7012	2 219
3	0.0340244	0.0334509	41.881	2 385
4	0.0334828	0.0341164	35.8616	2 473

Tabla 7.5: Resultados ajuste mediante ICP de la torre de las gallinas usando puntos clave. En la primera columna aparece el número de cada iteración en la segunda y tercera las distancias medias antes y después de cada iteración respectivamente. En la cuarta se incluye el tiempo en segundos que ha tardado cada iteración y en última columna el número de puntos clave de la primera toma con “correspondencia” en la segunda.

ponder el hueco en la pared que se encuentra a la izquierda y ha cuadrar las paredes de la habitación. Recordamos que el alineado obtenido no es el más refinado, ya que en caso de quererlo deberíamos usar todo el modelo. Finalmente, destacamos que en el último paso, la distancia ha aumentado tras el procedimiento lo que puede parecer contradictorio. Sin embargo, en ocasiones se ha visto en la práctica que el método cuando está cerca de la solución óptima empieza a oscilar. Además, en

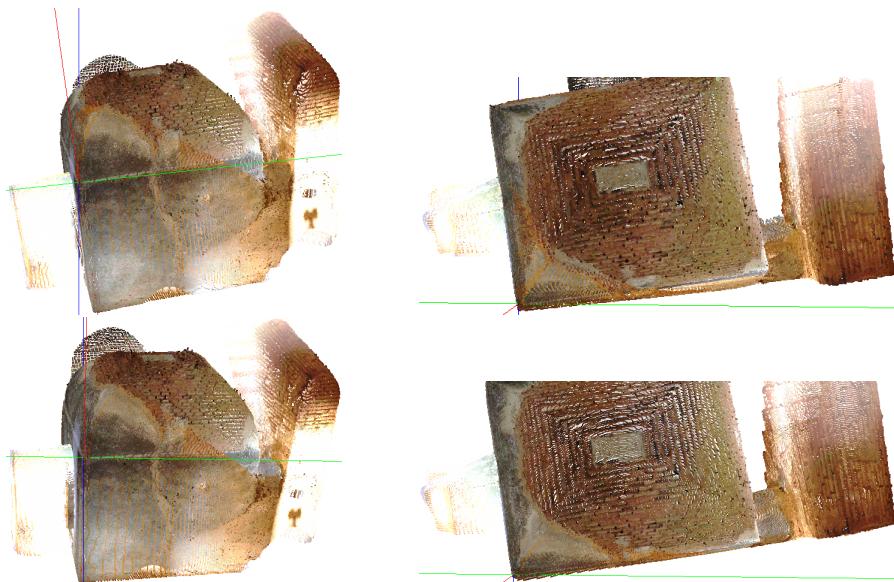


Figura 7.14: Resultado de la primera iteración (fila superior) y de la cuarta iteración (fila inferior).

las pruebas que estamos realizado no tenemos en cuenta un criterio de parada en función de la variación de la distancia media obtenida. Por último, también vemos que el número de puntos clave a los que se le aplica el método no es el mismo en cada caso. Por ello, aunque no es el caso, entre una iteración y otra no es raro que aumente la distancia media. Por último, en lo relativo al número de puntos clave, vemos que ha ido aumentando en cada paso por lo que las tomas están cada vez “más cerca”.

7.4. Sensibilidad y uso en prealineado

En la sección anterior, hemos usado los puntos clave para alinear dos tomas. En el ejemplo, tras el prealineado dichas tomas estaban “bien” posicionadas, es decir, lo suficientemente cerca la una de la otra para asegurar en todo lo posible que el algoritmo funcione correctamente. Nos hacemos ahora las siguientes preguntas: ¿cómo de sensible es el método propuesto a la alineación inicial? o ¿se podría usar para el propio prealineado?. La primera no es una cuestión baladí, ya que, como se ha indicado anteriormente, es un paso crucial. Si a dos nubes de puntos, se le aplica el procedimiento general ICP no podemos asegurar que si repetimos el proceso con otro prealineado también funcione. Con esta sección, pretendemos determinar cómo de bien posicionadas deben de estar las tomas. Cuanto menos sensible sea, menos nos debemos de preocupar del mismo e incluso si le afecta poco, podríamos usarlo sin necesidad de prealineado. Destacar que el objetivo de esta sección no es el análisis del tiempo que tarda en ejecutarse el algoritmo por lo que no se incluirán dichas tablas como en anteriores ejemplos.

De este modo, volvemos a las tomas que usamos anteriormente y se muestran

en las figuras 7.8 y 7.9. Si aplicamos el procedimiento a las muestras tomadas directamente desde el escáner sin ningún prealineado obtenemos la secuencia de la figura 7.15. Destacar que se ha quitado el límite de longitud máxima para considerar el mínimo. De otro modo, los puntos clave más alejados no se tendrán en cuenta.

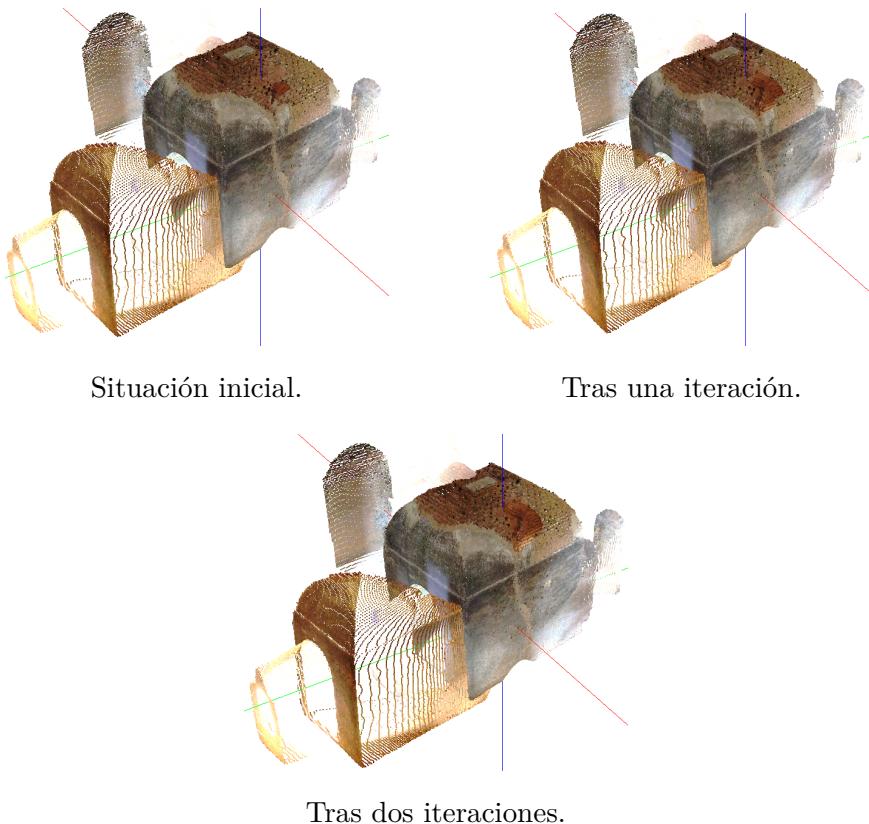


Figura 7.15: Ejemplo de sensibilidad. Aplicación del algoritmo ICP sin prealineado.

Vemos que el resultado no ha sido satisfactorio al no haber prácticamente cambios en el modelo. Esta situación puede deberse en cierta medida a que en la primera nube, se centran los puntos clave en la zona de la escalera y en la segunda, en la cúpula. Observamos que dichas zonas están juntas por lo que intenta ajustar esas zonas. Utilizamos ahora un prealineado no demasiado exacto para ver cómo se comporta el método. A partir de ahora, sí utilizaremos la cota para aceptar el mínimo. Esta se basará en un valor aproximado a la distancia entre los terceros vértices del prealineado ya que de ese modo obtenemos cierta referencia de la distancia que manejamos. Claramente, este valor debe ser mayor ya que las tomas están más separadas lo que también conllevará que se incremente el tiempo de cómputo. Los resultados aparecen en las figuras 7.16, 7.17 y 7.18.

Vemos, que como en el anterior, el modelo apenas cambia tras aplicarle el algo-



Figura 7.16: Situación inicial. Prealineado no exacto.



Figura 7.17: Tras una iteración. Prealineado no exacto.



Figura 7.18: Tras dos iteraciones. Prealineado no exacto.

ritmo. Finalmente, realizamos el proceso con un prealineado un poco más fino que el anterior. Este proceso aparece reflejado en las figuras 7.19, 7.20 y 7.21.

En este caso, hemos tardado 6 iteraciones en conseguir un resultado que podríamos considerar satisfactorio. No es tan preciso como en la sección anterior cuando el prealineado era mejor, pero teniendo en cuenta la situación inicial lo podemos considerar como aceptable. Por ejemplo, si nos fijamos en la evolución de las fotos de la izquierda, observamos cómo la esquina inferior se va acercando cada vez

más entre ambas tomas hasta estar bastante cercana en ambos casos

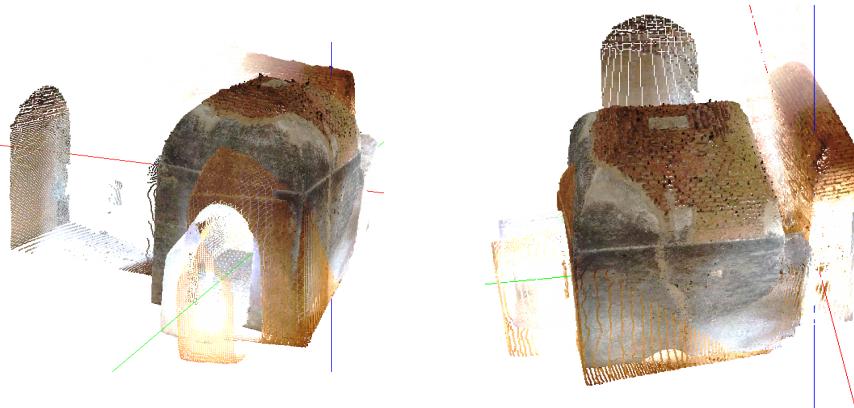


Figura 7.19: Situación inicial. Tomas mejor prealineadas.

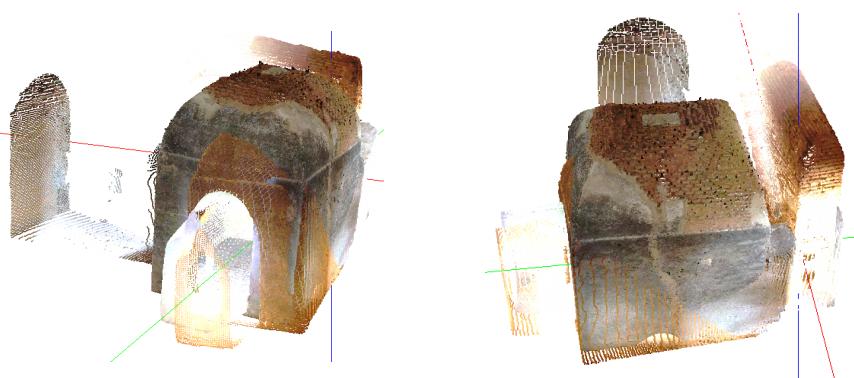


Figura 7.20: Tras una iteración. Tomas mejor prealineadas.

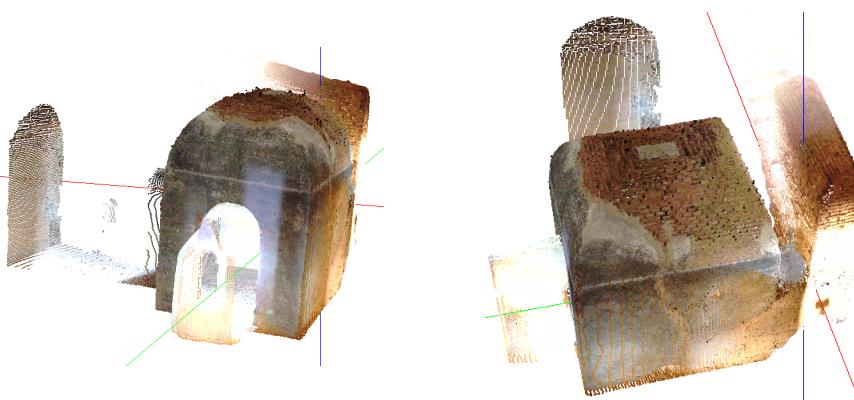


Figura 7.21: Tras seis iteraciones. Tomas mejor prealineadas.

7.5. Conclusiones

Concluimos el estudio del algoritmo ICP con una sección de conclusiones acerca de todo lo que hemos visto hasta ahora. En primer lugar, gracias a este procedimiento hemos conseguido un mecanismo para alinear dos nubes correspondientes a un mismo objeto. Para poder comprenderlo hemos tenido que realizar un estudio previo acerca de los cuaternios. Sin embargo, este algoritmo tiene una convergencia local lo que significa que ambas tomas deben estar lo suficientemente “cerca” una de otra para obtener una solución adecuada. Esta situación hace necesario el uso de un prealineado inicial e incluso haciendo uso de él puede que no consigamos un resultado satisfactorio. Otro inconveniente que presenta es la complejidad cuadrática en tiempo. Esto implica que hasta en modelos simples, tarde bastante tiempo en completarse cada una de las iteraciones. Por ello, para intentar acelerar el proceso hemos introducido unos descriptores para cada uno de los puntos basándonos en la variación de la normal. Esto ha sido posible gracias a la forma que tiene el escáner usado para la realización de tomas de darnos los resultados. De esta forma hemos detectado zonas significativas en ambas tomas y realizado el mecanismo de alineación entre ellas. Hemos podido comprobar que los resultados han sido satisfactorios teniendo en cuenta tanto la simplificación del modelo para la obtención de las normales como el hecho de usar una parte de cada una de las nubes de puntos en el procedimiento.

RANSAC: algoritmo de consenso

En esta sección estudiaremos otro de los algoritmos más utilizados para la alineación de dos nubes de puntos. Se trata del algoritmo *Random Sample Consensus* (RANSAC). El método ICP que acabamos de ver podríamos decir que es específico para la solución de nuestro problema, es decir, no se utiliza más allá de la situación en la que lo hemos estudiado. Por su parte, el algoritmo RANSAC es más genérico y utilizado en otras tareas diferentes. Su finalidad es la de ajustar parámetros de un modelo matemático que viene determinado por un conjunto de observaciones. Nosotros lo utilizaremos para la detección de superficies planas en nuestros modelos que a su vez las utilizaremos para la obtención de sus puntos intersección. Esos puntos que podríamos considerarlos como clave, siguiendo la nomenclatura utilizada, serán la base para el alineado. Destacar que es considerado un algoritmo robusto incluso bajo la existencia de gran cantidad de valores atípicos y que sigue un paradigma de hipótesis-y-prueba.

8.1. Algoritmo

Este algoritmo fue propuesto por A. Fischler y Robert C. Bolles en [14]. Este será el artículo que seguiremos como referencia en lo que respecta a la explicación matemática de la veracidad del método. Como hemos dicho en la introducción, podemos considerar que es un algoritmo del tipo hipótesis-y-prueba. A diferencia de otros algoritmos, no tiene en cuenta todo el conjunto, sino que toma el menor número posible de puntos necesarios para ajustar el modelo y comprueba si efectivamente esos puntos determinan el modelo que queremos. En nuestro caso, el modelo que deseamos obtener es un plano.

El algoritmo consta de las siguientes etapas:

1. Tomar de manera aleatoria el mínimo número de puntos necesarios que define el modelo.
2. Con una tolerancia ε , contar el número de puntos que distan del modelo definido en el punto anterior.
3. Repetir 1 y 2 un total de N veces y quedarse con el que contenga una mayor cantidad de puntos.

De este modo, podemos considerar que el algoritmo necesita, por ahora, tres parámetros. El primero de ellos, el mínimo número de puntos para definir el modelo. Por ejemplo, si queremos obtener una recta debemos tomar 2 puntos mientras que si queremos obtener un plano debemos coger 3. También tenemos que decidir una tolerancia ε que depende, en cierto modo, de la nube de puntos que tenemos entre manos y del resultado final que queremos. Finalmente, hay que decidir un N que determina el número de veces que repetimos el proceso. Su cálculo se explicará en siguientes secciones pero destacamos ahora que debe de ser lo suficientemente grande para asegurar que este proceso aleatorio conseguirá nuestro objetivo.

Antes de continuar, veamos un ejemplo sencillo en dos dimensiones donde explicaremos cómo funciona este procedimiento. Supongamos que tenemos la nube de la figura 8.1.

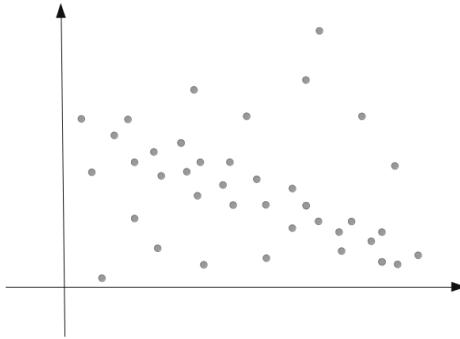


Figura 8.1: Nube de puntos.

Queremos aplicarle el algoritmo RANSAC para obtener una recta dentro de ella. Para ello, primero cogemos dos puntos aleatorios del modelo, calculamos al recta que pasa por ellos, y obtenemos los puntos que distan de la misma una distancia menor que ε . Supongamos que la situación es la que se muestra en la figura 8.2.

Si contamos el número de puntos que se ajustan al modelo, vemos que hay un total de 8. Repetimos el proceso y, en una iteración determinada, se obtiene el modelo de la figura 8.3.

En esta ocasión, un total de 27 puntos se ajustan al modelo. Si no se detectara en el resto de iteraciones un ajuste mejor, este sería el modelo que nos devolvería al algoritmo.

8.2. Número de iteraciones

Uno de los puntos clave del algoritmo es asegurarse de que el número de iteraciones N sea lo suficientemente grande para obtener puntos que ajusten correctamente

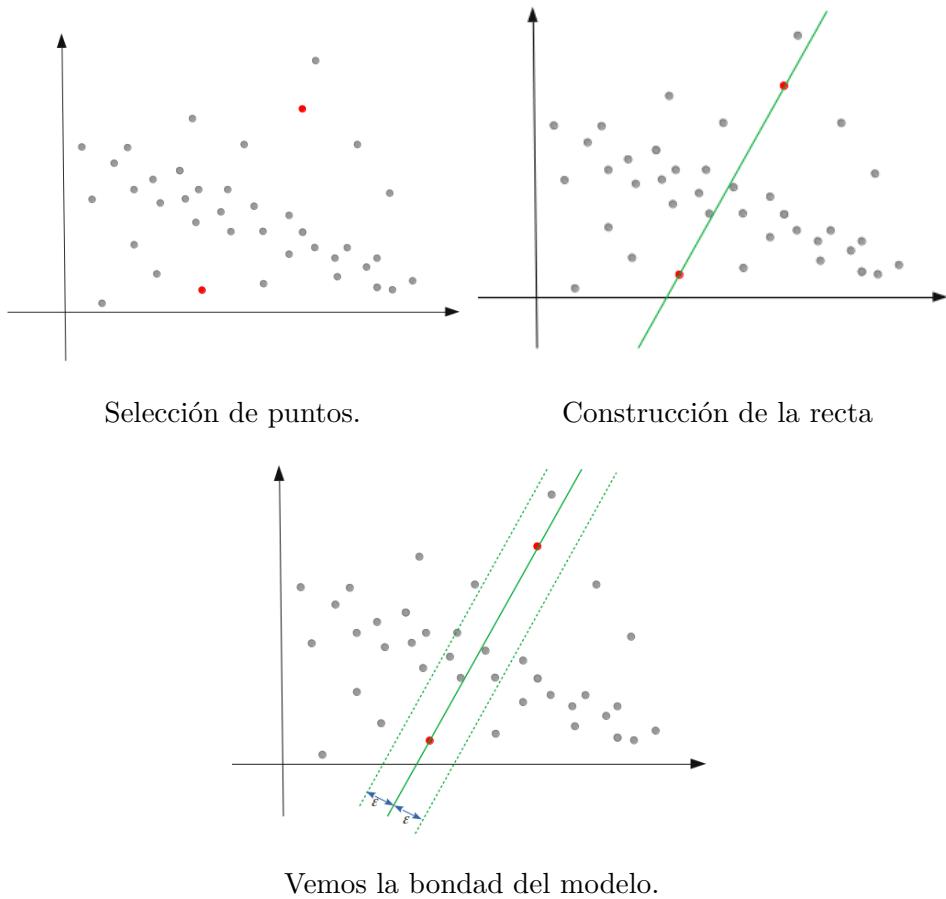


Figura 8.2: Primer paso del algoritmo RANSAC.

el modelo que queremos. De este modo, sea p la probabilidad que al menos uno de los conjuntos tomados no contiene valores atípicos y $1 - p$ de que todos los conjuntos contienen al menos un valor atípico. Notamos por u , a la probabilidad de que un valor sea correcto y, por lo tanto, $v = 1 - u$ es la probabilidad de ser un valor atípico. Sea m el número de puntos mínimo para definir el modelo: dos para una recta, tres para un plano, Así, la probabilidad de escoger todos los puntos correctos en una iteración es u^m y la de escoger alguno incorrecto es $1 - u^m$. Si hacemos un total de N iteraciones, la probabilidad de escoger siempre un valor atípico es $(1 - u^m)^N$. Pero esto no es más que

$$1 - p = (1 - u^m)^N.$$

Si despejamos el valor de N ,

$$N = \frac{\log(1 - p)}{\log(1 - u^m)} = \frac{\log(1 - p)}{\log(1 - (1 - v)^m)}. \quad (8.1)$$

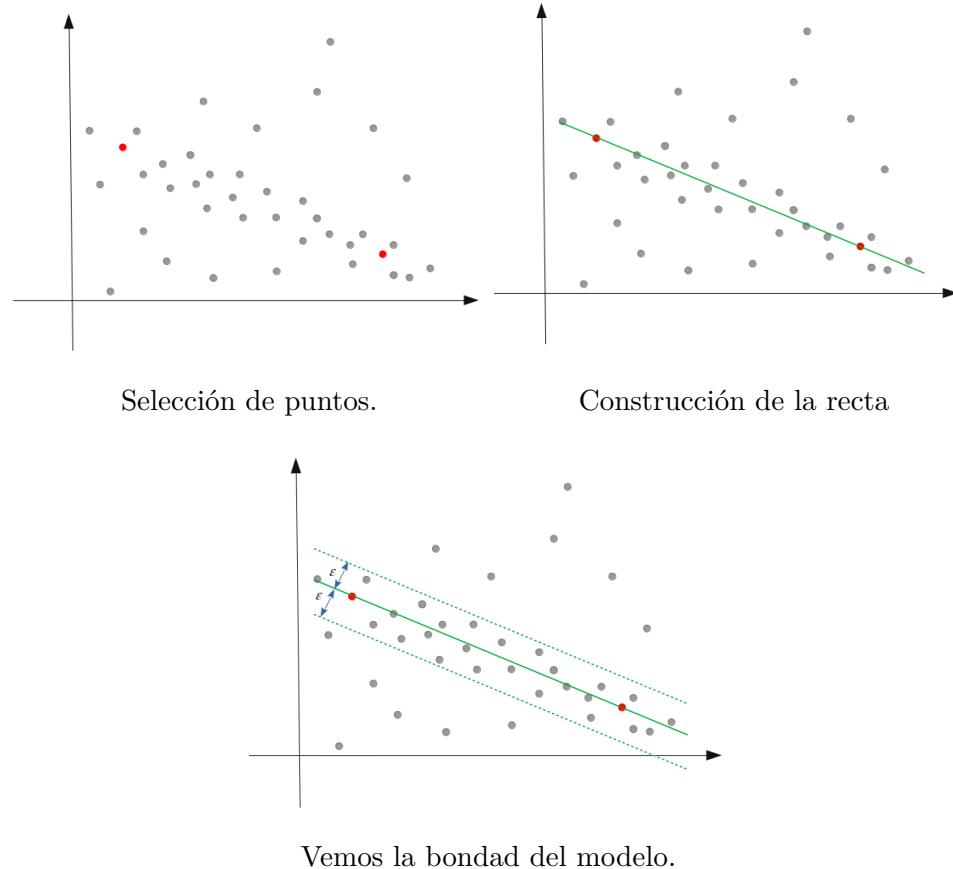


Figura 8.3: Paso del algoritmo RANSAC.

El valor obtenido no tiene que ser obligatoriamente un número natural por lo que se toma el primero entero que supere o iguale dicho valor. La probabilidad p es entrada del algoritmo y debe ser elevado. También u (o v) son parámetros que se eligen y que dependerán en cierto modo, del modelo con el que se está trabajando. La tabla 8.1 recoge diferentes valores de N en función de p , v y m .

8.3. Ejemplos sencillos

En esta sección, veremos cómo actúa al algoritmo en dos ejemplos concretos: el del pie y en el de la torre. En ambos casos usaremos los modelos simplificados tal y como hemos visto en la sección 7.1. Esto no debería de ser una restricción ya que la densidad total del modelo no varía y los planos se mantienen. Para cada, utilizaremos diferente valores de los parámetros y veremos cómo de buenos son los resultados obtenidos. Al ser modelos en los que buscamos planos el valor de m es 3. Destacamos que para el cálculo de la distancia entre el modelo y cada punto se ha hecho mediante la función distancia con signo (en valor absoluto).

	m	v			
		0.05	0.2	0.5	0.8
$p = 0,99$	2	2	5	17	113
	3	3	7	35	574
$p = 0,9$	2	1	3	9	57
	3	2	4	18	287
$p = 0,8$	2	1	2	6	40
	3	1	3	13	201

Tabla 8.1: Número de iteraciones para diferentes valores de p (probabilidad), m (número de puntos para definir el modelo: 2 recta y 3 plano) y v probabilidad de ser un valor atípico.

8.3.1. Pie

Se ha tomado como nivel de tolerancia $\varepsilon = 0,05$ ya que ese valor depende sobre todo del modelo que estamos trabajando. El modelo consta de un total de 34,120 puntos. En la tabla 8.2 y en la figura 8.4 están los resultados que nos proporciona.

Prueba	p	v	Tiempo (seg.)	Iteraciones
(a)	0.5	0.4	0.0521721	3
(b)	0.5	0.3	0.0260945	2
(c)	0.7	0.3	0.055964	3
(d)	0.9	0.3	0.116656	6
(e)	0.9	0.5	0.356769	18
(f)	0.9	0.7	1.7125	85

Tabla 8.2: Resultados de aplicar el algoritmo RANSAC una vez a una toma del pie. La primera columna identifica la prueba (ver figura 8.4), la segunda y tercera los valores de p y v escogidos y las dos últimas el tiempo empleado en el cálculo del plano y las iteraciones que se han realizado.

Vemos que al ser un proceso aleatorio los resultados que se han obtenido son dispares pero en este caso, en un número “bajo” de iteraciones hemos visto que es capaz de detectar el plano que contiene el modelo.

8.3.2. Torre

Realizamos diferentes pruebas con nuestro modelo de la torre que consta de 431 254 puntos. En esta ocasión se ha optado por una tolerancia de $\varepsilon = 0,03$. Los resultados se recogen en la tabla 8.3 y en la figura 8.5.

Vemos que nuevamente se han podido obtener resultados buenos en relativamente pocas iteraciones.

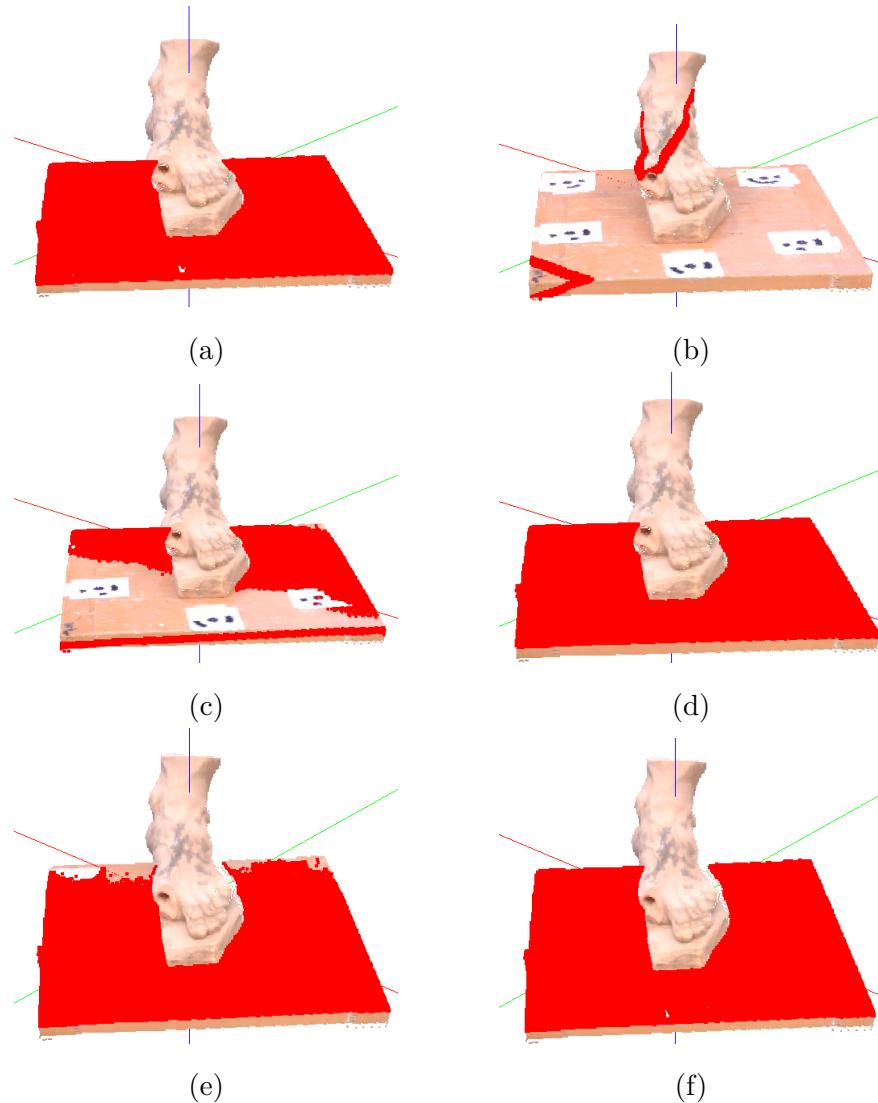


Figura 8.4: Resultados de aplicar RANSAC en el ejemplo del pie para diferentes parámetros (ver tabla 8.2).

8.4. Cálculo de los puntos de intersección

En esta sección, vamos a obtener mediante el algoritmo RANSAC un punto que sea intersección de tres planos. Para ello, tomamos el ejemplo de la torre. En primer lugar, veamos cómo calculamos dicha intersección. Lo único que tenemos que hacer, dado un conjunto de puntos que hemos obtenido como plano, es tomar una normal del mismo. Esta normal puede ser, por ejemplo, la misma que hemos usado para el cálculo de las distancias mediante la función distancia con signo que la podemos ir almacenando para evitar su cálculo posterior. Notamos a la normal obtenida para el paso i como $N_i = (N_{i1}, N_{i2}, N_{i3}) \in \mathbb{R}^3$. De este modo, cada punto

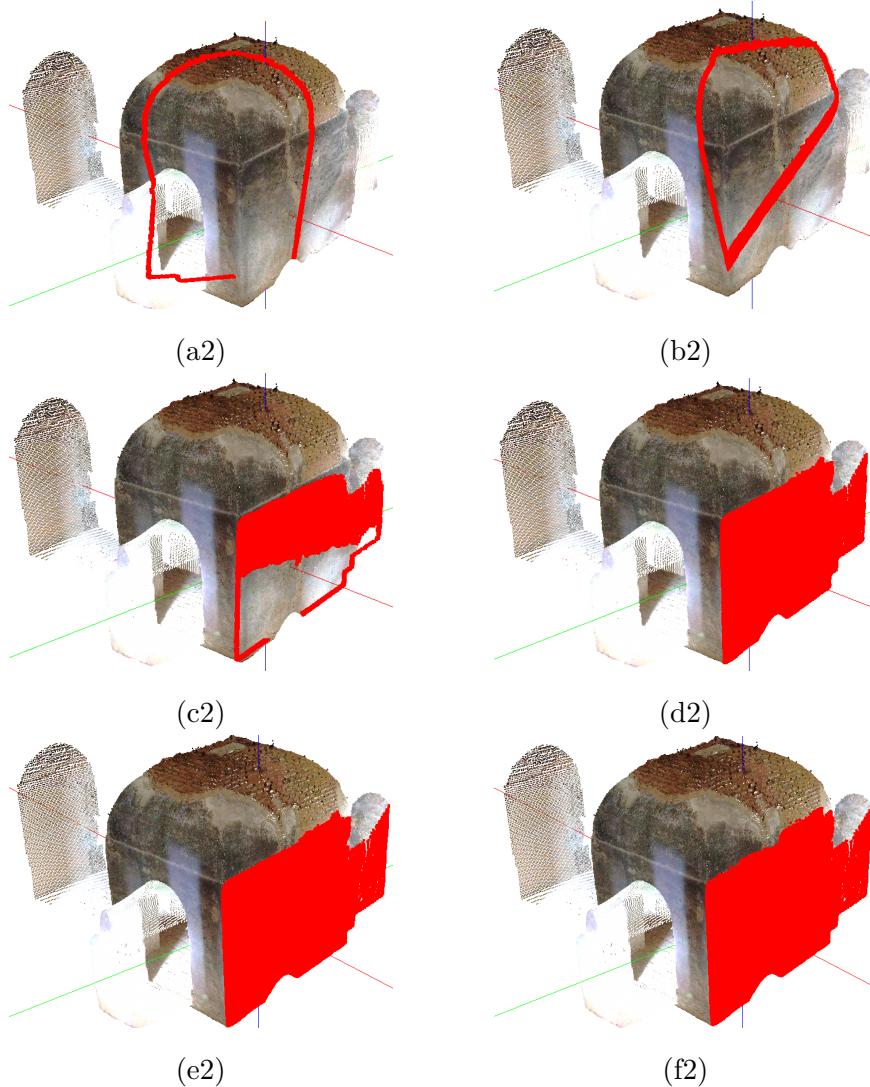


Figura 8.5: Resultados de aplicar RANSAC en el ejemplo de la torre para diferentes parámetros (ver tabla 8.3).

(x, y, z) del conjunto debe cumplir la ecuación:

$$N_{i1}x + N_{i2}y + N_{i3}z = d_i.$$

Finalmente, solo nos queda calcular d_i , que se puede hacer de manera sencilla sustituyendo el valor de (x, y, z) por un punto conocido del conjunto obtenido. Hemos calculado la ecuación del plano de una manera sencilla. Ahora solo nos queda calcular la intersección de tres planos. Para ello, tenemos que resolver el sistema para las iteraciones i, j, k :

Prueba	p	v	Tiempo (seg.)	Iteraciones
(a2)	0.5	0.4	0.484033	3
(b2)	0.7	0.4	0.961606	5
(c2)	0.7	0.6	4.27183	19
(d2)	0.8	0.6	5.74916	25
(e2)	0.8	0.7	14.0042	59
(f2)	0.85	0.8	55.9862	237

Tabla 8.3: Resultados de aplicar el algoritmo RANSAC una vez a una toma de la torre. La primera columna identifica la prueba (ver figura 8.4), la segunda y tercera los valores de p y v escogidos y las dos últimas el tiempo empleado en el cálculo del plano y las iteraciones que se han realizado.

$$\begin{aligned} \left. \begin{array}{l} N_{i1}x + N_{i2}y + N_{i3}z = d_i \\ N_{j1}x + N_{j2}y + N_{j3}z = d_j \\ N_{k1}x + N_{k2}y + N_{k3}z = d_k \end{array} \right\} \\ \Updownarrow \\ \begin{pmatrix} N_{i1} & N_{i2} & N_{i3} \\ N_{j1} & N_{j2} & N_{j3} \\ N_{k1} & N_{k2} & N_{k3} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} d_i \\ d_j \\ d_k \end{pmatrix} \\ \Updownarrow \\ A \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \mathbf{d}. \end{aligned}$$

Si el sistema es compatible determinado, lo que significa que la intersección está formada solo por un punto, podemos calcular A^{-1} y obtenemos el punto de intersección como:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = A^{-1}\mathbf{d}.$$

Veamos este caso en el modelo real. Primero, obtenemos tres planos de la nube de puntos. Usamos el algoritmo con valor $p = 0.8$, $v = 0.8$ y $\varepsilon = 0.03$. Los planos que se han obtenido se muestran en la siguiente figura 8.6.

Los valores que se han obtenido han sido los siguientes:

- $N_1 = (0,987981, 0,154204, 0,0107157)$, $d_1 = -0,50992$.
- $N_2 = (-0,0192558, 0,0458887, 0,998761)$, $d_2 = -1,13957$.
- $N_3 = (0,147295, -0,987795, 0,0506549)$, $d_3 = -1,34888$.

Claramente, el sistema que forman es compatible determinado. Si lo resolvemos tal y como se ha explicado anteriormente obtenemos el punto $(-0,690391, 1,20058, -1,20946)$ que se corresponde con el marcado en la figura 8.7.

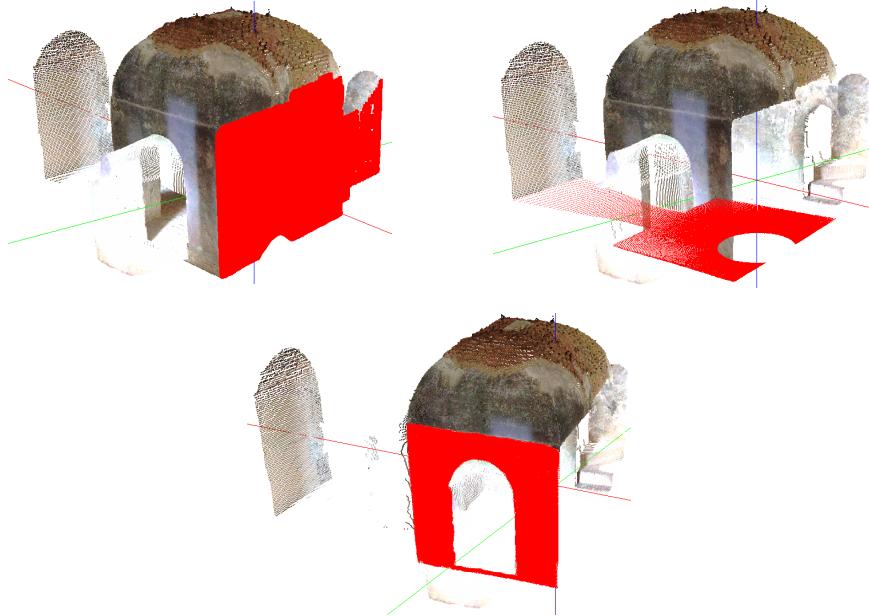


Figura 8.6: Planos detectados iterativamente en el ejemplo de la torre de las gallinas de la Alhambra. Los parámetros son $p = 0,8$, $v = 0,8$ y $\varepsilon = 0,03$.

8.5. Ejemplo despacho

El objetivo de esta sección es probar el comportamiento del algoritmo en un ejemplo más complejo y del que podemos obtener un mayor número de planos. En concreto, se trata de dos tomas de un despacho. Destacamos que nuevamente se ha realizado el proceso de simplificado de las mismas con un nivel de 4 y tras este proceso se han obtenido aproximadamente 650 000 puntos en cada toma. En la figura 8.8 se muestran ambas nubes de puntos.

En primer lugar, veamos los planos que se obtienen al ejecutar el algoritmo RANSAC en la primera de ellas. Los planos obtenidos se muestran en la figura 8.9. Posteriormente, en la tabla 8.4 aparece el tiempo empleado y el número de

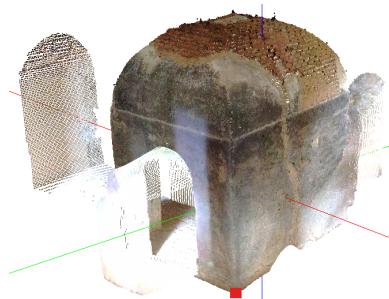


Figura 8.7: Punto de intersección.

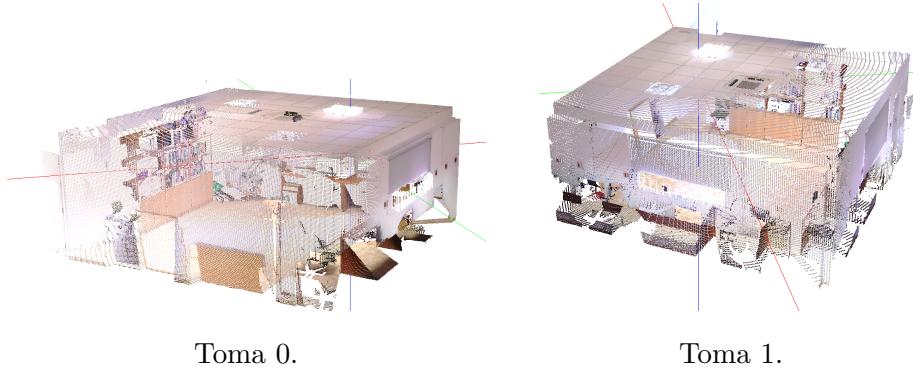


Figura 8.8: Tomas del laboratorio.

puntos detectados en cada plano. Finalmente, en la figura 8.10 están los puntos de intersección obtenidos.

Iteración	Tiempo (seg.)	Num. puntos
1	273.455	239 099
2	169.57	98 342
3	113.872	27 342
4	100.32	56 325
5	72.0483	46 067
6	64.7628	17 130

Tabla 8.4: Resultados de aplicar RANSAC en la toma 0 del laboratorio. En la primera columna aparece el número de la iteración, en la segunda el tiempo empleado en segundos y en la tercera el número de puntos que contiene cada plano detectado.

Repitamos el proceso con la otra toma. Los resultados se encuentras en la figuras 8.11 y 8.12 y en la tabla 8.5.

Una vez que hemos obtenido los puntos clave, lo que tenemos que hacer es agruparlos, es decir, ver la correspondencia entre ellos y calcular la transformación que lleva un conjunto en otro. Así, la idea es calcular tres correspondencias y aplicar un prealineado al igual que hicimos en la fase previa del algoritmo ICP. Nuevamente, utilizaremos los descriptores para obtener características significativas de los puntos y poder asignar unos con otros. Para ello, tomamos como referencia el artículo [15]. En él, propone varios parámetros posibles para formar parte del descriptor aunque el mejor de todos y el que usaremos es el vector formado por los ángulos de intersección de los planos en dicho punto. La explicación de usar este descriptor es que estos ángulos solo dependen de la geometría del modelo y no de otros factores como pueden ser el punto de vista. Una vez los hemos calculado, se obtiene una matriz con las distancias entre los ángulos. Ahora, aplicamos un algoritmo *greedy* o voraz para obtener las tres mejores correspondencias: vamos tomando en cada paso



Figura 8.9: De izquierda a derecha y de arriba a abajo los planos que se han ido detectando en la toma 0.

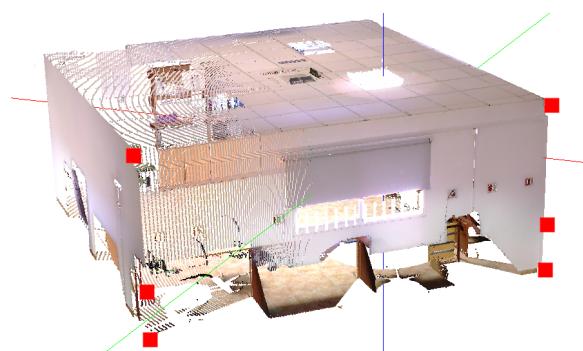


Figura 8.10: Intersecciones entre los planos en la toma 0.

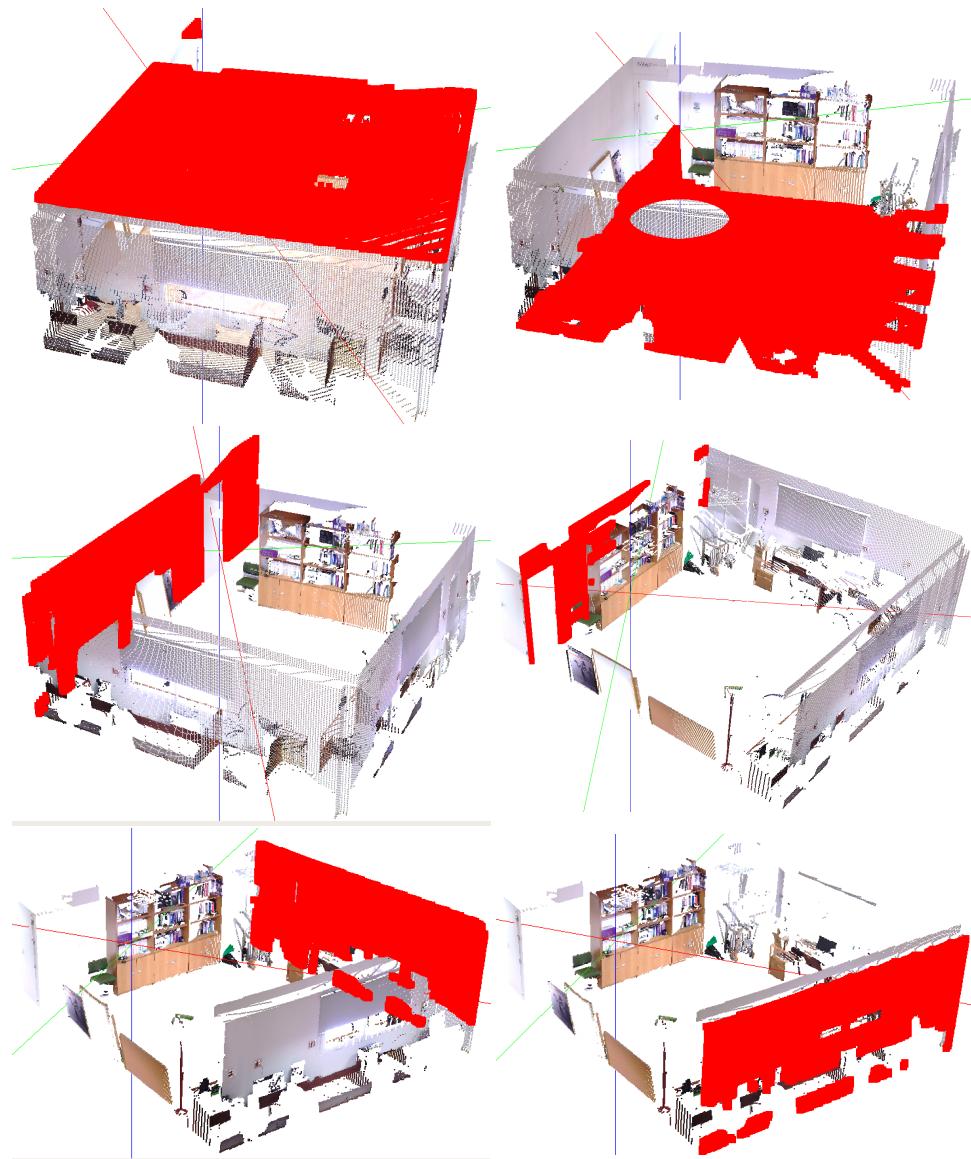


Figura 8.11: De izquierda a derecha y de arriba a abajo los planos que se han ido detectando en la toma 1.

la menor de las distancias siempre y cuando no se repitan puntos ya obtenidos. La razón de que no se puedan repetir los puntos es bastante simple, ya que, de otro modo no se podría realizar el prealineado. Por ello, el resultado obtenido se muestra en la figura 8.13.

Tal y como se aprecia en la figura, los resultados no han sido los esperados. ¿A qué se puede deber este resultado? En primer lugar, a la propia geometría del modelo. La habitación tiene sus paredes formando un ángulo recto entre ellas lo

Iteración	Tiempo (seg.)	Num. puntos
1	342.291	269 194
2	203.528	103 483
3	151.065	83 988
4	90.8403	31 849
5	87.0826	15 167
6	75.1206	12 961

Tabla 8.5: Resultados de aplicar RANSAC en la toma 1 del laboratorio. En la primera columna aparece el número de la iteración, en la segunda el tiempo empleado en segundos y en la tercera el número de puntos que contiene cada plano detectado.

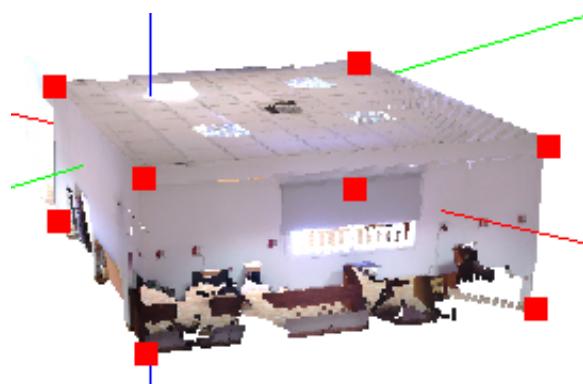


Figura 8.12: Intersecciones entre los planos en la toma 0.

que hace que todos los descriptores sean bastante parecidos. Sin embargo, como ya hemos dicho: no existe un algoritmo que funcione con todos los modelos. En segundo lugar, a los propios errores de medición a los que se suman los del cálculo de los planos. Esto hace que las pequeñas variaciones que pueda haber se disipen. También tenemos que tener en cuenta las propias apreciaciones realizadas en el artículo tomado como referencia. Indica que la única manera para obtener la solución óptima sería hacer un algoritmo exhaustivo, es decir, probar todas las posibilidades. Ello conllevaría un gasto de tiempo enorme para el cálculo de la alineación al tener que calcular en cada combinación posible la distancia mínima entre los modelos (que ya hemos visto que es muy costoso). Otras posibles soluciones serían, por ejemplo, a partir de esos puntos que el usuario los seleccione manualmente como en el prealineado de ICP. La ventaja frente a esa situación es, que al ser calculados los puntos automáticamente, se obtendría un mejor resultado que si el usuario lo hiciera de una manera visual. Otra opción sería, siempre y cuando se hubiesen obtenido los mismos puntos en ambos casos, calcular la distancia media solo entre los puntos obtenidos mediante RANSAC.

También influye en gran medida el número de planos obtenido durante el proceso. Según el artículo de referencia, el porcentaje de acierto con 15 planos no llega ni al 10 % mientras que unos 30 ya ronda el 80 %. Esto hace pensar que el algoritmo

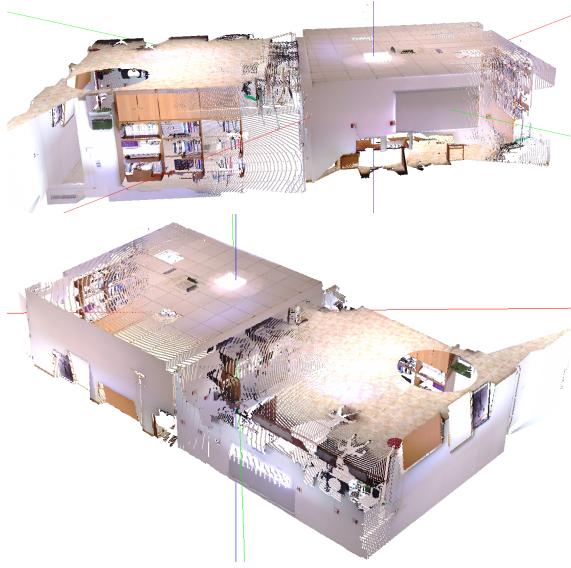


Figura 8.13: Resultado de la alineación.

será más indicado con modelos más complejos que el usado en nuestro caso.

8.6. Ejemplo específico

En la sección anterior acabamos de ver que el algoritmo propuesto no funcionaba correctamente en nuestro modelo. Como adelantamos, esto no significa que no se pueda utilizar en otras ocasiones en las que podamos disponer de un modelo adecuado. Incluso en [15], dice que en el ejemplo que propone no se realizan todas las correspondencias adecuadamente. Por ello, se ha creado un ejemplo *ad hoc* para demostrar que realmente funciona. En nuestro ejemplo ficticio tenemos los planos

$$\left\{ \begin{array}{l} x = 2 \\ y = 2 \\ z = 2 \\ -0,707106x + 0,707106z = -2 \end{array} \right.$$

que conforman una “toma”. La otra está formada por la rotación de $\frac{\pi}{2}$ alrededor del eje X, es decir,

$$\left\{ \begin{array}{l} x = 2 \\ z = 2 \\ -y = 2 \\ -0,707106x - 0,707106y = -2 \end{array} \right.$$

En la figura 8.14 vemos cómo quedan ambos conjuntos.

Los puntos de intersección detectados en el primer conjunto son: $(2, 2, 2)$, $(4,82843, 2, 2)$ y $(2, 2, -0,82843)$. En el segundo caso, son estos puntos tras aplicar la rotación in-

dicada anteriormente. Los puntos calculados se pueden observar en la figura 8.15

Si calculamos los ángulos que forman los planos en cada uno de los puntos obtenemos que en un caso es $(\frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2})$, en otro $(\frac{\pi}{4}, \frac{\pi}{2}, \frac{\pi}{2})$ y en el último $(\frac{\pi}{2}, \frac{\pi}{2}, \frac{3\pi}{4})$. En la otra toma estos valores no varían en los correspondientes puntos. Como vemos, ahora los valores son bastante diferentes entre sí y por ello el algoritmo los asigna correctamente y observamos el resultado en la figura 8.16.

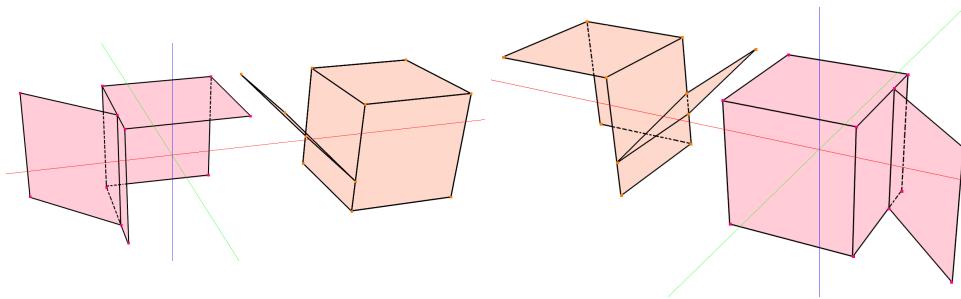


Figura 8.14: Planos de ejemplo. Notamos que en una toma se ha modificado la matriz de modelado¹ para poder visualizarlos por separado.

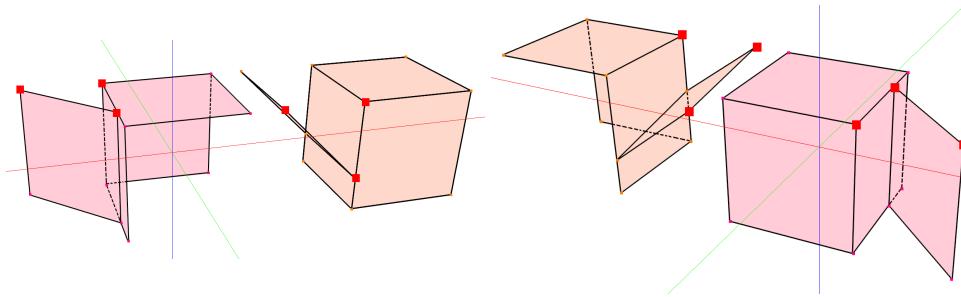


Figura 8.15: Puntos de intersección calculados en el ejemplo.

Concluimos que los resultados desfavorables obtenidos en la sección anterior se deben sobre todo a la simetría del modelo disponible y que en otros modelos sí es capaz de funcionar correctamente.

¹Esta matriz posiciona los puntos en su lugar en coordenadas de mundo.

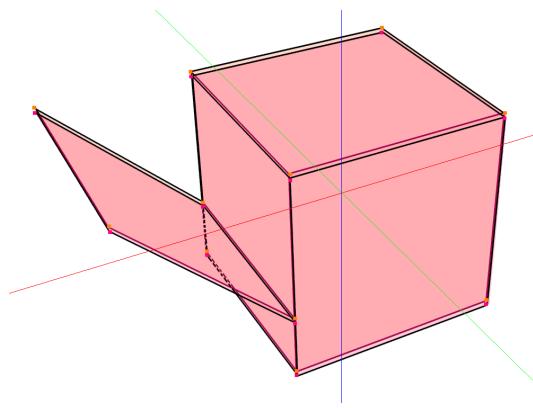


Figura 8.16: Resultado de la alineación. Se ha aplicado un leve cambio en una matriz de modelado para poder apreciar la superposición de ambos conjuntos.

Banco de pruebas

En esta última sección nos dedicaremos a explicar algunos aspectos relevantes acerca del banco de pruebas que se ha desarrollado. Como se ha indicado anteriormente, la finalidad de este trabajo fin de grado no era la del desarrollo de una aplicación que se pudiera usar para el proceso digitalizado 3D sino del estudio de los ventajas e inconvenientes de cada uno de los métodos que intervienen. Por ello, esta aplicación ha sido una herramienta más usada durante el proceso. Sin embargo, se ha considerado que merece tener cierta atención ya que se han implementado algunas características interesantes, especialmente las relacionadas con las nubes de puntos, así como aspectos importantes de la Informática Gráfica. Por eso, no se va a hacer un estudio detallado acerca de la aplicación pero sí se van a explicar aspectos generales de la misma.

Para el desarrollo del programa ha sido de gran ayuda la referencia [16]. En él se explica cómo desarrollar aplicaciones haciendo uso de la biblioteca *OpenGL* (*Open Graphic Library*) para el ámbito gráfico y *C++* como lenguaje de programación. Sin embargo, la ventaja la encontramos en que utiliza la herramienta *Qt*. La biblioteca gráfica sólo nos permite producir los gráficos. Por ello, necesitamos una interfaz de usuario que nos permita la interacción y esa es la razón de usar *Qt*. A modo aclaratorio, también es necesario el uso de *GLEW* (*OpenGL Extension Wrangler Library*) que nos permite una gestión más fácil de *OpenGL*. También es conveniente destacar que el sistema operativo en el que se ha llevado a cabo todo este trabajo ha sido *Ubuntu 18.04.4* sobre un ordenador con procesador *Intel Core i7-3537U* y tarjeta gráfica *GeForce GT 720M*.

Finalmente, notamos que la aplicación que se propone en [16] ha servido como base para nuestro estudio. Principalmente, para el manejo de los diferentes objetos y algunos mecanismos como la selección de un punto (*picking*). El código se ha ido completando y modificando según las necesidades que teníamos en cada etapa del desarrollo.

9.1. Estructura básica de la aplicación

Una vez se han explicado las herramientas con las que trabajamos, la aplicación base para la visualización consta de las siguientes clases:

- *main*: es el programa principal. Su única finalidad es mostrar la ventana de la aplicación.

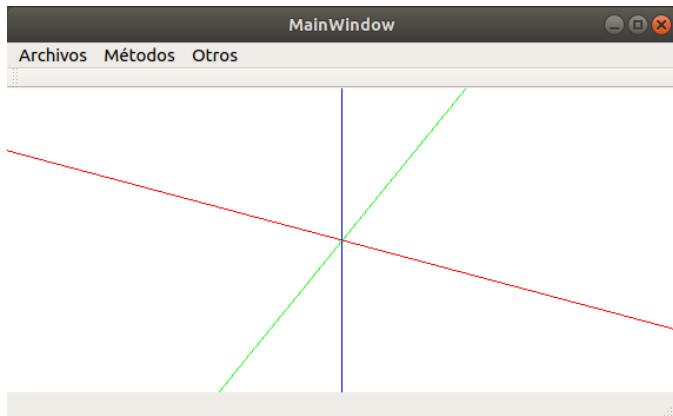


Figura 9.1: Interfaz de usuario del banco de pruebas programado.

- *mainwindow*: es la encargada de crear la interfaz de usuario. Deriva a su vez de la clase *QMainWindow*. En el archivo *mainwindow.ui* encontramos almacenada la información relativa a su configuración, principalmente, los menús con los que iremos activando cada una de las funciones necesarias.
- *_gl_widget*: clase derivada de *QOpenGLWidget* y que implementa las funciones para dibujar con *OpenGL*. Destacamos las funciones *initializeGL*, *resizeGL* y *paintGL* que sirven para iniciar *OpenGL*, actualizar el tamaño de la ventana y dibujar respectivamente. El código está preparado para visualizar los vértices, aristas o con relleno. Sin embargo, al trabajar nosotros con nubes de puntos, solo se visualizarán los mismos.

También se han programado los *shaders*. A modo aclaratorio, estos programas se ejecutan en la tarjeta gráfica del juego (no en la CPU) y que se aplican para transformar vértices, aplicar iluminación o crear algún tipo de efecto. Existen dos diferentes: *vertex shader* que se aplica para cada vértice y que transforma vértices, normales, calcula iluminación, etc. y *fragment shader* para cálculo de color de un fragmento (información para generar un píxel). En nuestro caso se han programado, por un lado, los dos *shaders* para la visualización normal de los objetos, y otro para la selección de puntos o *picking* que en la sección 9.2.

En cuanto al manejo de la cámara, se hace mediante teclado siendo posible tanto acercar o alejar la cámara y rotarla para observar mejor la imagen. La cámara se mueve alrededor de un esfera con centro el origen y no se ha añadido la posibilidad de cambiarlo. Por ello, es posible que haya objetos que podamos visualizar bien debido a su situación en el espacio. Esto se ha solucionado con la posibilidad de modificar la matriz de modelado de los mismos (mediante combinación de teclas). Esta matriz, como se ha indicado al final del capítulo anterior, se aplica a cada uno de los vértices del modelo y sirve para posicionarlos. Por ejemplo, cuando es la identidad, los puntos se mantienen en las mismas coordenadas que indican sus vértices. Sin embargo, si componemos transformaciones (en nuestro caso traslaciones pero también se puede hacer con cualquiera) a la misma conseguimos desplazar o escalar el objeto sin añadir carga extra. Esto se debe en el propio *vertex shader* se utiliza para calcular la posición de los puntos, junto las matrices de proyección y

vista (ver sección 9.3).

También relacionada con evitar demasiada carga de trabajo a la aplicación, nos encontramos con la manera de enviar los vértices a la tarjeta gráfica. Nos encontramos que estamos trabajando con una gran cantidad de puntos por lo que necesitamos que este trabajo se haga de manera rápida. Para solventar este inconveniente se usa el *envío en diferido* de los datos mediante los llamados *Vertex Buffer Object* (VBO). Estos son vectores que solo pueden ser usados por las tarjetas gráficas. Esto implica que los datos que necesitamos, vértices y colores, se encuentren en la GPU (*Graphics Processing Unit*) en vez de en memoria principal, acelerando el proceso de lectura de los mismos. Estos se han agrupado en un *Vertex Array Object* (VAO), que básicamente son estructuras para almacenar VBOs y que los activa automáticamente.

Dentro del mismo VBO almacenamos la información relativa a diversos objetos. Por ello, necesitamos saber en cada momento qué posición ocupa cada uno de esos objetos para poder, por ejemplo, visualizar algunos de ellos. Es ahora cuando entra en juego la clase *_object_management*. Esta clase contiene todos los objetos de la escena y se encarga del manejo de las posiciones de inicio dentro del vector y del tamaño de cada uno de los mismos.

9.1.1. Clase *_basic_object3D*

La clase que representa un objeto genérico es *_basic_object3D*. Encontramos por ejemplo los vértices, colores, normales, puntos claves, etc. Se ha ido modificando según las necesidades que teníamos en cada momento. Destacamos que de esta clase deriva otra, *_malla_ptx* que es la encargada de guardar y leer objetos que vienen en un archivo con formato PTX.

Destacamos que los puntos se mantienen en dos vectores y en una matriz. Un primer vector, que podríamos considerar auxiliar, con solo los vértices y que es el que se utiliza para llenar el VBO. El segundo y la matriz contienen elementos de una estructura que hemos definido y que se denomina *Punto*. Contiene tanto las coordenadas del vértice así como información relevante del mismo: posición dentro de la matriz (en el caso del vector) o del vector (en el caso de la matriz) y si el punto está activo o no. El hecho de hacerlo se debe a que cada uno de los casos es beneficioso para unas operaciones u otras. La matriz es la que nos indica la relación de vecindad entre cada uno de los puntos. Se usa, por ejemplo, para acelerar el cálculo de las normales ya que solo tenemos que acceder a las posiciones de la matriz que rodean al vértice actual y no es necesario buscarlas en el vector de vértices. Por otro lado, el vector es más intuitivo a la hora del manejo de vértices en general. En cuanto a si al punto se encuentra activo o no se refiere a si se ha borrado o no. Al realizar una toma, generalmente no se capta solamente el objeto deseado en sí, sino también parte de su entorno. Así, necesitamos indicar los puntos que no nos interesan (ver sección 9.3) y borrarlos. En vez de ir modificando el vector de vértices dejando solo los activos, lo que conllevaría una gran carga de trabajo al escribir constantemente en memoria, solo se marca si un vértice está activo o no. De este modo, una vez se tienen marcados los que se quieren visualizar, solo se

tiene que actualizar el vector que utiliza el VBO. En cuanto a los colores de cada vértice, tenemos también dos vectores como en el caso de los vértices: uno con todos los colores (tanto de vértices activos como no activos) y otro que se envía a la GPU.

Es interesante destacar las funciones para leer y guardar los archivos, sobre todo esta última. Cuando hemos modificado un objeto, ya sea porque hemos alterado su posición o hemos eliminado puntos, necesitamos mantener las relaciones de vecindad del conjunto que teníamos originalmente (para poder calcular las normales, por ejemplo). También, para el caso de que se hayan borrado puntos, el archivo que contiene el modelo modificado debería ocupar menos espacio. Por eso, para guardar, lo que hacemos es encontrar la mínima matriz que contiene a todos los vértices activos y solo guardar dichas filas y columnas. Es posible que esta submatriz contenga puntos que no estén activos. Como no es posible guardar dicha información en el formato PTX, que es con el tipo de datos que estamos trabajando, se ha optado por darle unas coordenadas a esos vértices muy alejadas. Así, al leer el modelo, vemos si la distancia de cada vértice al origen está dentro de una cota. Si no la cumple, marcamos el vértice como no activo y no se dibujaría, pero se seguiría manteniendo la coherencia espacial. Esta cota también la podemos utilizar para filtrar directamente puntos al cargar un archivo y ahorrarnos el eliminarlos manualmente (ver figura 9.2). Al cargar también tenemos en cuenta otro parámetro que nos aporta el formato PTX: el coseno del ángulo de incidencia en el plano. De este modo, solo nos quedamos con aquellos puntos que no estén muy oblicuos ya que eso implica mayor error en la toma de la muestra. Tanto estos valores de filtrado como los de los algoritmos se han pasado al programa mediante lectura de ficheros. Apuntar que el simplificado de las mallas que hemos mencionado en capítulos anteriores, se hace igual que la operación de guardar, solo aumentamos en cada iteración el incremento del índice para recorrer la matriz.

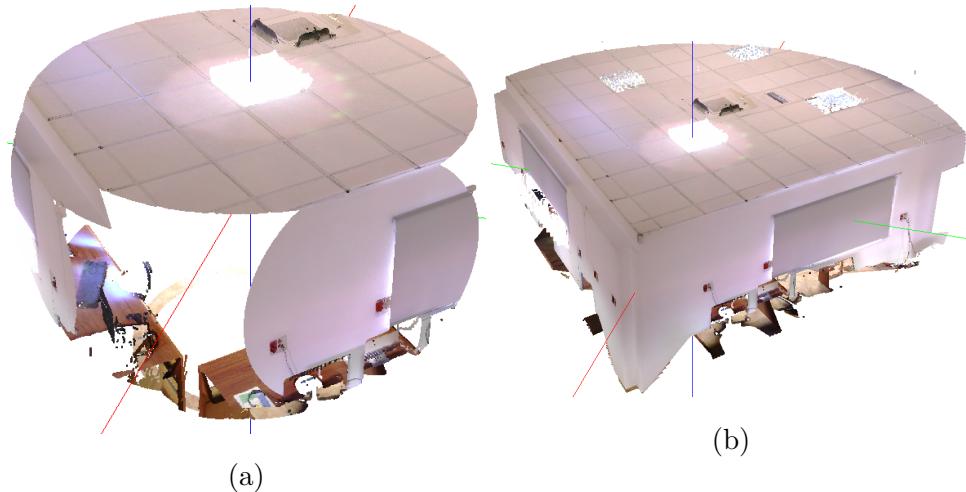


Figura 9.2: Ejemplo de filtrado según la distancia al escáner al cargar un modelo: (a) filtrado a 2 metros y (b) filtrado a 4 metros.

Esta clase proporciona los funciones con los métodos de alineado que hemos

estudiado. Destacamos que los procesos se han ido dividiendo en etapas. Esto se debe a que lo que nos interesaba era conocer el proceso en sí y no tanto tener un único mecanismo para hacerlo completo. Por eso, en ICP tenemos tanto la función de prealineado junto con otra para ejecutar solamente un paso del algoritmo (aunque también hay una función para ejecutar varias iteraciones con condición de parada la original y un número máximo de iteraciones si fuese necesario). Por su parte, el proceso de RANSAC se ha dividido a su vez en el cálculo de planos, poder cargarlos a partir de un archivo, cálculo de intersecciones entre los mismos, etc. Esta división también facilita las tareas de depuración ya que tenemos cada tarea por separado así como poder reproducir los resultados una vez tenemos calculados correctamente algunos datos.

9.2. Picking

Esta sección junto con la siguiente podemos considerar que son más interesantes dentro del ámbito de la Informática Gráfica. El *picking* consiste en seleccionar un único elemento de la escena. En nuestro aplicación, queremos seleccionar los vértices para el proceso de prealineado. Este proceso debe ser tanto rápido como interactivo. Por ello, una de las soluciones más usadas para solucionar este problema es la llamada *selección por color*. Este método consiste en tener un identificador único para cada elemento y convertirlo de manera única en un color. De este modo, podemos pasar de identificador a color y viceversa sin ningún tipo de problemas. Así, lo único que hay que hacer es dibujar el objeto con estos colores modificados, obtener el color de la posición que se ha seleccionado y recuperar el identificador. Como un color en RGB (*Red Blue Green*) se representa mediante tres bytes, tenemos un total de 16 777 216 de combinaciones. A esta cantidad habría que quitarle la combinación del blanco que se utiliza para indicar que no se ha seleccionado nada.

La relación entre identificador y color es de la siguiente manera:

- Identificadores de 0 a 255 van a la componente azul.
- Identificadores de 256 a 65 535 van a la componente verde.
- Identificadores de 65 535 a 16 777 215 van a la componente roja.

Esta conversión se hace fácilmente mediante máscaras y rotaciones de los bits. Ya tenemos la manera de asociar colores e identificadores, ahora, ¿cuál es el identificador para cada vértice que vamos a usar?. Usaremos la posición que ocupa cada vértice en el vector del VBO y que nos lo aporta directamente la directiva *gl_PrimitiveID* del *fragment shader*. Como se ha mencionado en 9.1 necesitamos por tanto otro programa *shader* diferente: en el *vertex* solo calculamos la posición en el mundo de cada vértice y en el *fragment* hacemos la conversión de identificador a color.

Volviendo al proceso, hemos comentado que tenemos que dibujar el objeto con los nuevos colores asignados. Esto no lo podemos hacer en la pantalla ya que ahí queremos visualizar el objeto normal. Si se hiciese de ese modo se observaría un parpadeo. Se utiliza entonces un *framebuffer*, esto es, una zona de memoria donde se dibuja la imagen y se hacen otras operaciones como el cálculo de las profundidades (algoritmo *z-buffer*). El *framebuffer* principal es el único que permite mostrar por la

pantalla por lo que debemos crear uno propio, asignándole tanto zona de memoria para los colores como otra para el *z-buffer*. Finalmente, solo falta obtener el color del pixel seleccionado mediante la función *glReadPixels*. En la figura 9.3, encontramos un ejemplo de visualización normal y otras con los colores para el *picking*.

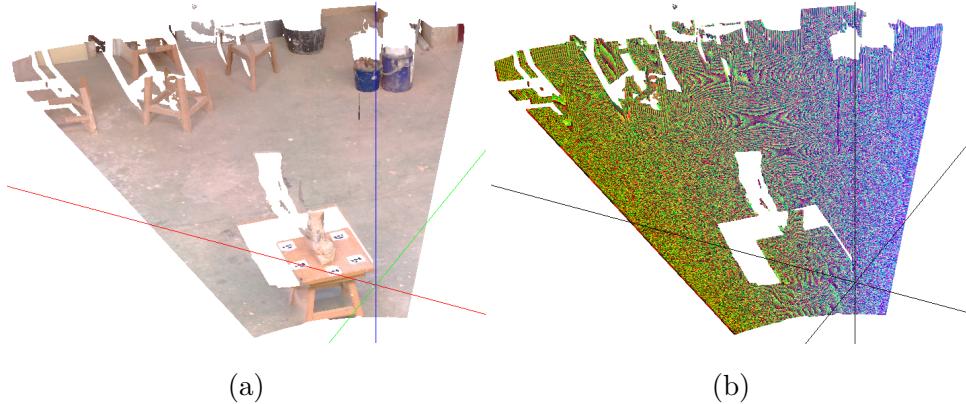


Figura 9.3: Visualización con los dos programas *shader*: (a) visualización normal y (b) visualización con la asignación de colores según identificador para el picking

9.3. Selección mediante rectángulo

Otra de las funciones a nombrar y que se han incluido en el banco de pruebas es la posibilidad de seleccionar puntos mediante un rectángulo. Esto es clave para eliminar las partes que no nos interesan de las tomas. Uno de los aspectos que involucra a este tema ya se ha explicado y es mantener la información de si un punto está activo o no para no elevar la carga de trabajo. El segundo aspecto a destacar lo tratamos aquí es la propia detección de puntos que están dentro del rectángulo y que se ha abordado de manera diferente al *picking*.

En primer lugar, necesitamos dibujar el propio rectángulo que indica la selección. Para ello se ha usado un objeto la clase *QRubberBand* que proporciona *Qt*. Además, ya tiene un método que nos proporciona si un determinado elemento está dentro del mismo o no. ¿Cuál es el aspecto interesante? Se trata del tipo de coordenadas que recibe como entrada dicha función y que veremos a continuación.

Para poder dibujar correctamente la figura en pantalla, hay que tener en cuenta no solo el lugar del objeto en el mundo virtual sino también la posición de la cámara, modificaciones en el propio objeto, etc. Esto ya se ha tratado al hablar de las traslaciones del objeto mediante modificaciones de la matriz de modelado, por ejemplo. Así, el flujo para obtener el lugar de la pantalla donde se proyecta cada vértice es el siguiente:

1. Los vértices vienen dados en coordenadas de objeto que es la posición según el marco de referencia del propio objeto.

2. Primero aplicamos la matriz de modelado para obtener las coordenadas del mundo que son comunes a toda la escena.
3. A continuación, mediante la matriz de vista, se obtienen las coordenadas de ojo y que se puede ver como la transformación que nos sitúa en el marco de referencia de la cámara.
4. Una vez hecho esto, necesitamos recortar la imagen indicando la región que queremos que sea visible así como el tipo de proyección que queremos, en nuestro caso principalmente con perspectiva. Esto se hace mediante la matriz de proyección.

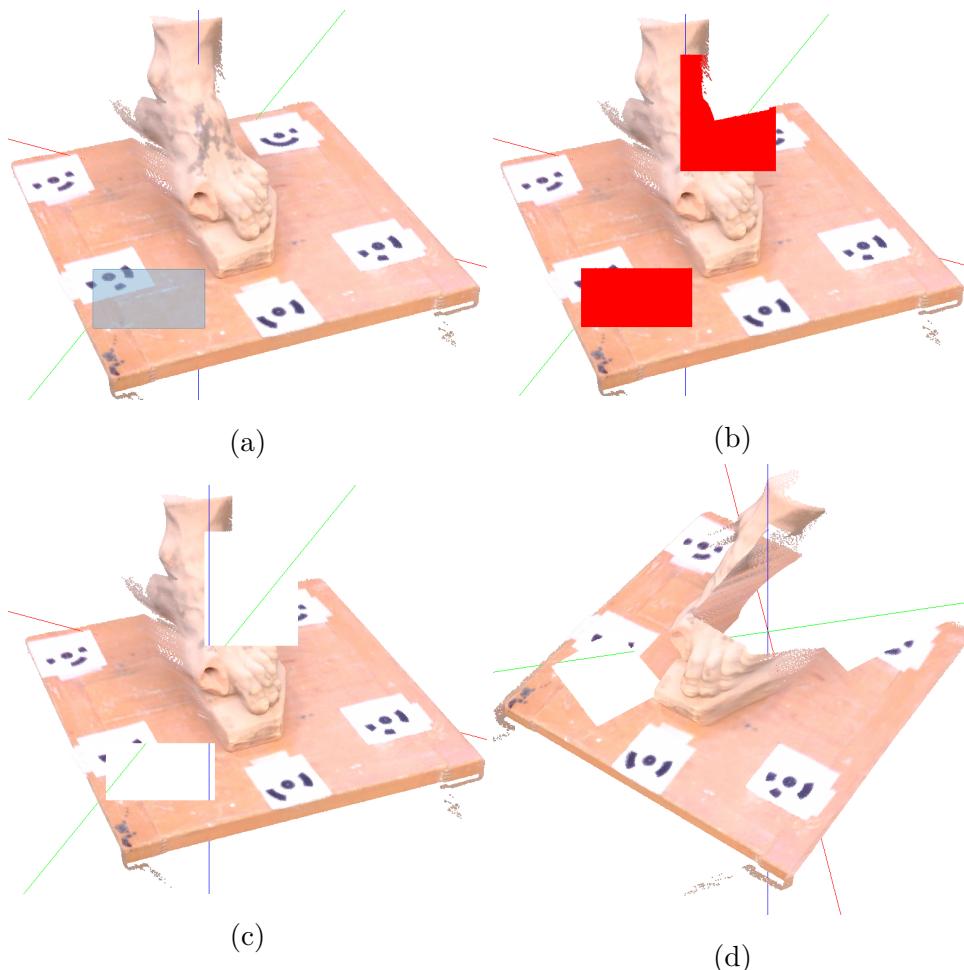


Figura 9.4: Borrado de puntos: (a) rectángulo de selección, (b) zona a borrar con posibilidad de mantener los seleccionados anteriores, (c) y (d) modelo tras borrar los puntos seleccionados.

5. A continuación se obtienen las coordenadas normalizadas de dispositivo, para que las coordenadas del punto se encuentren entre $[-1, 1]$. Esto se hace

mediante la división de un cuarto parámetro que obtenemos directamente del paso anterior.

6. Por último, se tienen las coordenadas de dispositivo mediante la transformación lineal de llevar el intervalo $[-1, 1]$ al tamaño y posición del *viewport* dentro de la ventana actual.

Una vez explicado el proceso, notamos que las matrices de modelado y de vista son relativamente fáciles de calcular: la primera porque es resultado de transformaciones que he hemos ido aplicando y para la segunda hay que tener en cuenta que la cámara viene dada en nuestro caso en coordenadas esféricas. En cuanto a la matriz de proyección, la podemos obtener directamente pasando los parámetros del área que queremos visualizar. Así, la función *contains* de nuestro rectángulo recibe las coordenadas de dispositivo y que son necesarias calcularlas manualmente. En la figura 9.4 vemos un ejemplo de selección y borrado de un conjunto de puntos.

En definitiva, aunque el proyecto se haya centrado en otros aspectos, gracias al desarrollo de este banco de pruebas se ha podido conocer nuevos entornos de desarrollo. También se han profundizado en otros aspectos relacionados con la informática gráfica que en mayor o menor medida ya se habían estudiado anteriormente pero se ha considerado que eran lo suficientemente relevantes para volver a tratarlos. Además, se han debido de solucionar problemas, como guardar modelos modificados, que aunque no son relevantes en las conclusiones obtenidos, son de suma importancia a la hora de poder trabajar.

Conclusiones y vías futuras

Tanto en el ámbito matemático como informático, los objetivos que nos marcamos en la propuesta inicial se han alcanzado satisfactoriamente. En el primero de ellos, incluso se ha realizado una incursión no prevista en la teoría minimax de manos de los teoremas de la alternativa. Ello ha permitido obtener una visión completa de las técnicas y aplicaciones de los teoremas de la alternativa.

En el caso del de digitalizado 3D, se ha realizado una sólida introducción con los procedimientos básicos existentes. El estudio hecho en este trabajo se podría completar posteriormente con varias vías debido a lo amplitud del tema. Sería posible plantear otra serie de mejoras a los algoritmos como por ejemplo, el uso de *voxels* para tener una mejor división espacial de la nube de puntos y acelerar el proceso del cálculo del punto más cercano. Otro posible camino a seguir sería estudiar algoritmos para la resolución de las otras dos etapas del proceso: fusión y triangulación. Finalmente, también se podría mejorar el banco de pruebas con la finalidad de que sea un *software* funcional para el público en general.

Bibliografía

- [1] J.M. Borwein, A.S. Lewis, *Convex analysis and nonlinear optimization: theory and examples*, Second Edition, CMS Books in Mathematics/Ourages Mathématiques de la SMC 3, Springer, New York, 2006.
- [2] R.J. Elliott, R.E. Kopp, *Mathematics of financial markets*, Second Edition, Springer Finance, New York, 2005.
- [3] G. Giorgi, A. Guerraggio, J. Thierfelder, *Mathematics of optimization: smooth and nonsmooth case*, Elsevier Science B.V., Amsterdam, 2004.
- [4] H. König, *Über dans von Neumannsche minimax-theorem*, Archiv der Mathematik 19(1968), 482-487.
- [5] H. König, *Sublinear functionals and conical measures*, Archiv der Mathematik 77(2001), 56-64.
- [6] S. Mazur, W. Orlicz, *Sur les espaces métriques linéaires II*, Studia Mathematica 13(1953), 137-179.
- [7] E. Schechter, *Handbook of analysis and its foundations*, Academic Press, Inc., San Diego, CA, 1999.
- [8] S. Simons, *From Hahn-Banach to monotonicity*, 2nd edition, Lectures notes in Mathematics 1693, Springer, New York, 2008.
- [9] Gary KL Tam, Zhi-Quan Cheng, Yu-Kun Lai, Frank C Langbein, Yonghuai Liu, David Marshall, Ralph R Martin, Xian-Fang Sun, Paul L Rosin, *Registration of 3D point clouds and meshes: A survey from rigid to nonrigid*, IEEE transactions on visualization and computer graphics, 2012.
- [10] Yan-Bin Jia, *Quaternions and rotations*, Com S 477/577, 2013. Notes
- [11] Jernej Barbic, *Quaternions and Rotations*, CSCI 520 Computer Animation and Simulation, University of Southern California.
- [12] Paul J.Besl y Neil D.McKay, *A method for registration of 3d-shapes*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1992.
- [13] Richard Hartley y Andrew Zisserman, *Multiple view geometry in computer vision*, Cambridge university press, 2003.
- [14] Martin A Fischler y Robert C Bolles, *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*, Communications of the ACM, 24(6):381–395, 1981.

- [15] PW Theiler, K Schindler, et al, *Automatic registration of terrestrial laser scanner point clouds using natural planar surfaces*, ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, 3:173–178, 2012.
- [16] Domingo Martín Perandrés, *Informática Gráfica con OpenGL 4*, 2018.
- [17] Qt Documentation, <https://doc.qt.io/qt-5/reference-overview.html>
- [18] Eigen C++ library documentation, <http://eigen.tuxfamily.org/dox/>