



UNIVERSIDAD DE CHILE  
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS  
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ALINEACIÓN AUTOMÁTICA DE MODELOS DENTALES 3D PARA  
PREVISUALIZACIÓN DE TRATAMIENTOS DE ORTODONCIA

MEMORIA PARA OPTAR AL TÍTULO DE  
INGENIERO CIVIL EN COMPUTACIÓN

MARIO CÉSAR CORNEJO CARRASCO

PROFESOR GUÍA:  
BENJAMÍN BUSTOS CÁRDENAS

MIEMBROS DE LA COMISIÓN:  
JAVIER BUSTOS JIMÉNEZ  
CLAUDIO GUTIÉRREZ GALLARDO

SANTIAGO DE CHILE  
2019

## RESUMEN

Wizz es una empresa que ofrece tratamientos de ortodoncia con fines estéticos por medio de alineadores transparentes. Los potenciales pacientes de estos tratamientos pueden agendar una sesión de escaneo dental 3D gratuita con la promesa de la entrega de un *Tratamiento Virtual*, el cual también es gratuito. Este *Tratamiento Virtual* será enviado por correo electrónico al potencial paciente luego de un par de días después de ocurrido el escaneo.

Un *Tratamiento Virtual*, para fines de esta empresa, hace referencia a los modelos dentales 3D digitales que evidencian el cambio que tendrá la sonrisa del paciente al someterse al tratamiento de ortodoncia pasando por todos los pasos intermedios. El *Tratamiento Virtual* por tanto tiene doble funcionalidad: primero sirve como una forma de que el paciente pueda ver el resultado final del tratamiento y con esto motivarlo a la compra. La segunda funcionalidad es que estos modelos dentales 3d sirven como moldes para fabricar los alineadores por medio de impresión 3D, esto en caso de que el tratamiento sea comprado. A este conjunto de alineadores se le llama *Tratamiento Físico*.

La fabricación de un *Tratamiento Virtual* requiere aproximadamente una hora de tiempo de trabajo de un operador. Este tiempo es bastante alto considerando que mucha de la gente a la que se le fabrica este tratamiento luego no compra el producto.

Se propone como nuevo concepto el *Tratamiento Virtual Preliminar* el cual es fabricado en un tiempo mucho menor por tratarse de un proceso semiautomático y que no será impreso físicamente en el futuro por lo que hay más tolerancia a las imperfecciones. Este tratamiento tiene como única finalidad motivar al paciente a comprar, por lo que debe ser entregado mientras dure la sesión de escaneo. El *Tratamiento Virtual* como tal se reservará solo para las personas que efectúen la compra del tratamiento, ya que es necesario para la fabricación del *Tratamiento Físico*.

La siguiente memoria describe el diseño e implementación de un algoritmo que automatiza el proceso de la alineación dental para hacer entrega del *Tratamiento Virtual Preliminar* en el tiempo requerido, integrándose en los sistemas actuales de Wizz.

El algoritmo propuesto utiliza como herramienta la regresión a una curva, y la obtención de la posición neutra de los dientes a través de aplicar el algoritmo de ICP con una biblioteca de dientes previamente alineados. Luego los dientes se posicionan en la curva de tal manera de no dejar espacios ni colisiones.

Los resultados obtenidos son satisfactorios, el algoritmo proporciona un alineamiento dental que disminuye los movimientos que debe hacer posteriormente el operador en comparación al proceso anterior que se hacía completamente manual, y por otra parte el proceso en su totalidad esta dentro de los rangos de tiempo requeridos.

*A mis padres, que pusieron mi educación siempre como primera prioridad*

# Agradecimientos

Primero que nada agradezco a mis padres Mario y Liliana, sin ellos este viaje de la educación superior no hubiera sido posible. Siempre me dieron la confianza y libertad para mantenerme en este camino a pesar de lo difícil que se puso en algunos momentos, haciéndome ver que este proceso era importante, pero que no debía verse como un requisito impuesto por la sociedad o ellos, sino como un proceso de satisfacción e inversión personal.

Agradezco a mis amigos del colegio con los cuales nunca he perdido contacto y que han sido esenciales para no vivir pegado al computador. A los amigos que hice en la universidad, Alexis Acevedo y Jaime Capponi, que siempre estuvieron ahí, dispuestos a una conversación para compartir lo bueno y lo malo de Beauchef.

Agradezco a los profesores del DCC que aparte de enseñar su ramo lograron transmitirme el gusto por la computación, y la belleza que existía en el buen software y en las buenas soluciones, en particular a mi profesor guía Benjamín Bustos.

Agradezco a Javier Liberman por la confianza para realizar este y otros proyectos, y en general a todos los trabajadores de Wizz que siempre estuvieron abiertos a darme apoyo y feedback sobre las soluciones de software que creamos en conjunto.

# Tabla de Contenido

<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.1.1. Tratamientos de Ortodoncia por medio de alineadores transparentes . . . . .	1
1.1.2. Wizz . . . . .	2
1.1.3. Modelo de negocios de Wizz . . . . .	2
1.1.4. Problema y oportunidad . . . . .	4
1.2. Objetivos . . . . .	7
1.2.1. Objetivo general . . . . .	7
1.2.2. Objetivos específicos . . . . .	7
1.3. Metodología . . . . .	7
<b>2. Conceptos Básicos</b>	<b>9</b>
2.1. Django . . . . .	9
2.2. Three.js . . . . .	10
2.2.1. Estructura de un Objeto 3D . . . . .	11
2.2.2. Sobre los ejes de coordenadas . . . . .	11
2.3. Arquitectura de los Sistemas de Wizz . . . . .	12
2.4. Piezas dentales . . . . .	14
2.5. Proceso para crear un Tratamiento Virtual en Wizz . . . . .	14
2.5.1. Segmentación . . . . .	14
2.5.2. Edición (alineación) . . . . .	17
2.6. Slack . . . . .	18
2.7. Iterative Closest Point . . . . .	18
2.8. Ajuste de curvas por medio del método de mínimos cuadrados . . . . .	19
2.8.1. Ajuste de curvas . . . . .	19
2.8.2. Método de los mínimos cuadrados . . . . .	20
<b>3. Algoritmo Propuesto</b>	<b>21</b>
3.1. Idea general . . . . .	21
3.2. Sobre la implementación del algoritmo . . . . .	22
3.3. Control de versiones . . . . .	22
3.4. Dientes a alinear por el algoritmo . . . . .	22
3.5. Obtención de una Curva . . . . .	24
3.5.1. Raycast . . . . .	24
3.5.2. Puntos seleccionados para efectuar el ajuste a una curva . . . . .	24
3.5.3. Deducción de una ecuación . . . . .	25

3.6.	Posición Neutra del diente . . . . .	27
3.6.1.	Obtención manual de dientes en posición neutra . . . . .	27
3.6.2.	Aplicación de ICP . . . . .	28
3.7.	Posicionamiento de los Dientes en la Curva . . . . .	29
3.7.1.	Modificación del Algoritmo de Colisión . . . . .	29
3.7.2.	Calculo de las medidas límites del Diente . . . . .	31
3.7.3.	Posicionamiento . . . . .	31
3.7.4.	Escalamiento de la curva . . . . .	32
3.8.	Optimizaciones . . . . .	34
3.8.1.	ICP en paralelo . . . . .	34
<b>4.</b>	<b>Integración del Algoritmo al flujo de Wizz</b>	<b>36</b>
<b>5.</b>	<b>Evaluación Experimental</b>	<b>40</b>
5.1.	Pruebas del algoritmo de alineación . . . . .	40
5.2.	Correctitud de la alineación alcanzada . . . . .	44
5.3.	Pruebas del proceso completo de fabricación de Tratamiento Virtual Preliminar	46
<b>6.</b>	<b>Conclusiones y Trabajo Futuro</b>	<b>47</b>
	<b>Bibliografía</b>	<b>49</b>



# Capítulo 1

## Introducción

### 1.1. Motivación

#### 1.1.1. Tratamientos de Ortodoncia por medio de alineadores transparentes

Los tratamientos de ortodoncia por medio de alineadores transparentes se han hecho más populares los últimos años. Estos tratamientos presentan varias ventajas con respecto al tratamiento tradicional por medio de brackets. Una de las ventajas más importantes es que al ser transparentes y estar apegados a la dentadura del paciente, estos no se notan, por lo que se trata de un tratamiento más estético. Además pueden retirarse para lavarse los dientes y para comer, lo que les da una ventaja con respecto a la higiene. A pesar de que el paciente puede sacárselos, es recomendable que los use por lo menos 22 horas al día.



Figura 1.1: Brackets tradicionales y Alineadores transparentes

Los alineadores son placas de plástico transparente hechas exactamente a la medida de los dientes del paciente, de manera que cubren los dientes de forma casi perfecta. Sin embargo en la placa los dientes cuentan con un pequeño desplazamiento (décimas de milímetro). Este desplazamiento provoca que los dientes del paciente experimenten una fuerza por parte del alineador, por lo que los dientes del paciente se mueven progresivamente adaptándose a la forma del alineador.

Un tratamiento de ortodoncia por medio de alineadores transparentes consta por tanto de un set de varios alineadores, los cuales tienen un orden específico para ser usados, en donde cada alineador se acerca más al estado deseado de los dientes. El paciente debe cambiar al alineador siguiente cada dos semanas. La cantidad de alineadores por consiguiente depende de



la gravedad del caso. Si el caso es leve puede bastar con 8 alineadores (4 meses de tratamiento) y si es más grave podría resultar hasta en un tratamiento de 30 alineadores. Esto depende de una relación entre el movimiento más grande que necesite un diente para llegar al estado final, y el límite de movimiento recomendable que debe hacer un alineador como máximo.

Como para progresar en el tratamiento el paciente solo necesita cambiar al alineador siguiente, esto le da a este tratamiento otra ventaja, ya que el paciente no necesita ir a controles periódicos con un odontólogo u ortodoncista. Mientras que en el tratamiento tradicional es necesario que el paciente asista a controles periódicos para que un ortodoncista ajuste los brackets (generalmente una vez al mes). Esto se traduce en una diferencia de costos considerable entre ambos tratamientos.

### **1.1.2. Wizz**

Wizz es una startup fundada el 2017 por un exalumno del Departamento de Ciencias de la Computación de la Universidad de Chile. Esta startup se dedica a la planificación, fabricación y venta de tratamientos de ortodoncia por medio de alineadores transparentes. Actualmente tiene en funcionamiento 5 Centros de Escaneo 3D en Santiago, Viña, Antofagasta, La Serena, Concepción, Bogotá y Buenos Aires y venden más de 100 tratamientos al mes.

Todo el software utilizado tanto en sistemas de información como también en el diseño de tratamientos, es software propietario perteneciente a Wizz y ha sido desarrollado dentro de la empresa. Esto es una de sus ventajas competitivas.

### **1.1.3. Modelo de negocios de Wizz**

Wizz ofrece tratamientos de ortodoncia completos a un precio fijo, independiente de la cantidad de alineadores que sean necesarios para tratar al paciente. El paciente que compra recibe una caja con todos los alineadores necesarios, y otros accesorios que hubiera adquirido según el plan escogido, como por ejemplo blanqueamientos dentales en gel. El tratamiento viene con la aprobación de un odontólogo. El tratamiento se monitorea de forma remota por medio de fotografías. Los tratamientos de Wizz están enfocados en resolver principalmente problemas estéticos, y no aseguran que resolverán problemas de mordida o apiñamiento severo. Esto se le hace saber al paciente antes de adquirir el producto y se le sugiere recurrir al tratamiento tradicional en caso de ser necesario.

Wizz cuenta con publicidad en redes sociales como Instagram y Facebook, además de aparecer en búsquedas de Google según ciertas palabras clave. La persona interesada que ve la publicidad puede acceder a una página web. En esta página web puede agendar una cita de forma gratuita en uno de los centros de escaneo de Wizz, sujeto a horarios y cupos determinados.

Una vez que el prospecto asiste a la cita agendada se escanea con un especialista y se obtienen los modelos 3D de su dentadura. Además se le explican los distintos productos que ofrece Wizz. Si el prospecto compra de forma inmediata, recibirá un descuento.

A toda persona que asiste al escaneo se le hace entrega por medio de un link a una página web de su *Tratamiento Virtual*. Este *Tratamiento Virtual* consiste en una visualización en 3D

del cambio progresivo que irá teniendo la dentadura del paciente a través de cada alineador, desde el inicio hasta el final del tratamiento. Como muestra la figura 1.2 el usuario puede avanzar en los pasos del tratamiento, donde cada paso corresponde al efecto que tendrá un alineador. Además el modelo puede rotarse con el mouse para ser visualizado en distintas perspectivas.



Figura 1.2: Vista del paciente de un Tratamiento Virtual

Este *Tratamiento Virtual* es una forma de motivar al paciente a comprar ya que puede ver una simulación del cambio estético que tendrá su sonrisa. El link puede ser entregado uno o dos días después del día del escaneo. Esto por el procesamiento de los modelos, diseño del tratamiento en sí, además de ser un proceso que pasa por más una persona y la existencia de una cola previa de pacientes. Sin embargo si no existiera esta cola el proceso podría tardar de 30 minutos a 1 hora.



Figura 1.3: Proceso de fabricación física de un alineador

Si la persona llega a comprar, se imprimen los mismos modelos del *Tratamiento Virtual* en una impresora 3D en resina. Luego una lámina de plástico transparente es termoformada al modelo macizo previamente impreso en resina, produciendo el alineador. A esto se le llama *Tratamiento Físico*. La figura 1.3 muestra la impresión 3D maciza del modelo en resina, luego el plástico transparente termoformado exactamente a este modelo y finalmente el plástico transparente por sí solo o alineador.

#### 1.1.4. Problema y oportunidad

Como se comentó en la sección anterior, a toda persona que asista al centro de escaneo se le elabora su *Tratamiento Virtual*. El problema es que mucha de esa gente termina sin adquirir el producto, por lo que se trata de horas de trabajo desperdiciado, considerando que este *Tratamiento Virtual* fue hecho con los mismos estándares que los tratamientos que luego son comprados e impresos físicamente.

La mayoría de la gente compra en el mismo centro de Wizz luego de haberse escaneado, aprovechando que se ofrece un descuento (el escaneador además gana una comisión si esto ocurre). La gente que sale del centro de escaneo sin comprar tiene la ventaja de ver su *Tratamiento Virtual* uno o dos días después de haber sido escaneada su dentadura, luego de que llegue el link al correo electrónico. Sin embargo no tiene un incentivo para comprar de forma inmediata, por lo que las ventas que ocurren fuera del centro de escaneo se dan en mucha menor cantidad.

Si se pudiera entonces elaborar un *Tratamiento Virtual Preliminar* que fuera hecho de forma más rápida (máximo 15 minutos), se podría hacer esperar al paciente este tiempo luego de haberse escaneado, para poder ver inmediatamente los resultados aproximados de su tratamiento. Con esto se pretende aumentar aún más las ventas realizadas en el centro de escaneo durante la misma cita, y a su vez en un futuro eliminar los Tratamientos Virtuales detallados hechos solo para incentivar al paciente a comprar. Estos se elaborarán solo para la gente que compra el producto, ya que forman parte del proceso para obtener el *Tratamiento Físico*, eliminando así muchas horas de trabajo innecesario.

Se propone entonces crear un sistema que automatice la alineación de los modelos dentales, de forma que pueda entregarse al paciente un *Tratamiento Virtual Preliminar* en la misma cita pactada para realizar el escaneo de la dentadura. El paciente estará al tanto de que se trata de un tratamiento hecho por un sistema computacional, y no tiene que ser estrictamente igual al tratamiento que recibirá en caso de comprar el producto.

La figura 1.4 muestra el flujo de negocio actual de Wizz. Se puede ver el problema principal que es que cada vez que ocurre un escaneo queda en cola inmediatamente la fabricación del *Tratamiento Virtual* el cual es costoso en horas de trabajo y no siempre concluye en una venta. Si la venta ocurre entonces se continúa con la fabricación del *Tratamiento Físico*.

La figura 1.5 muestra los procesos que se desean agregar en verde. En particular la fabricación del *Tratamiento Virtual Preliminar*, corresponde al trabajo que se desarrollará en esta memoria. Este proceso es en parte hecho por un sistema automático y por un operador.

Se puede ver como en este nuevo flujo la fabricación del *Tratamiento Virtual* detallado solo ocurre si es que ocurre la venta. Además se pretende aumentar las ventas en el centro de escaneo por medio de la muestra casi inmediata del *Tratamiento Virtual Preliminar*.

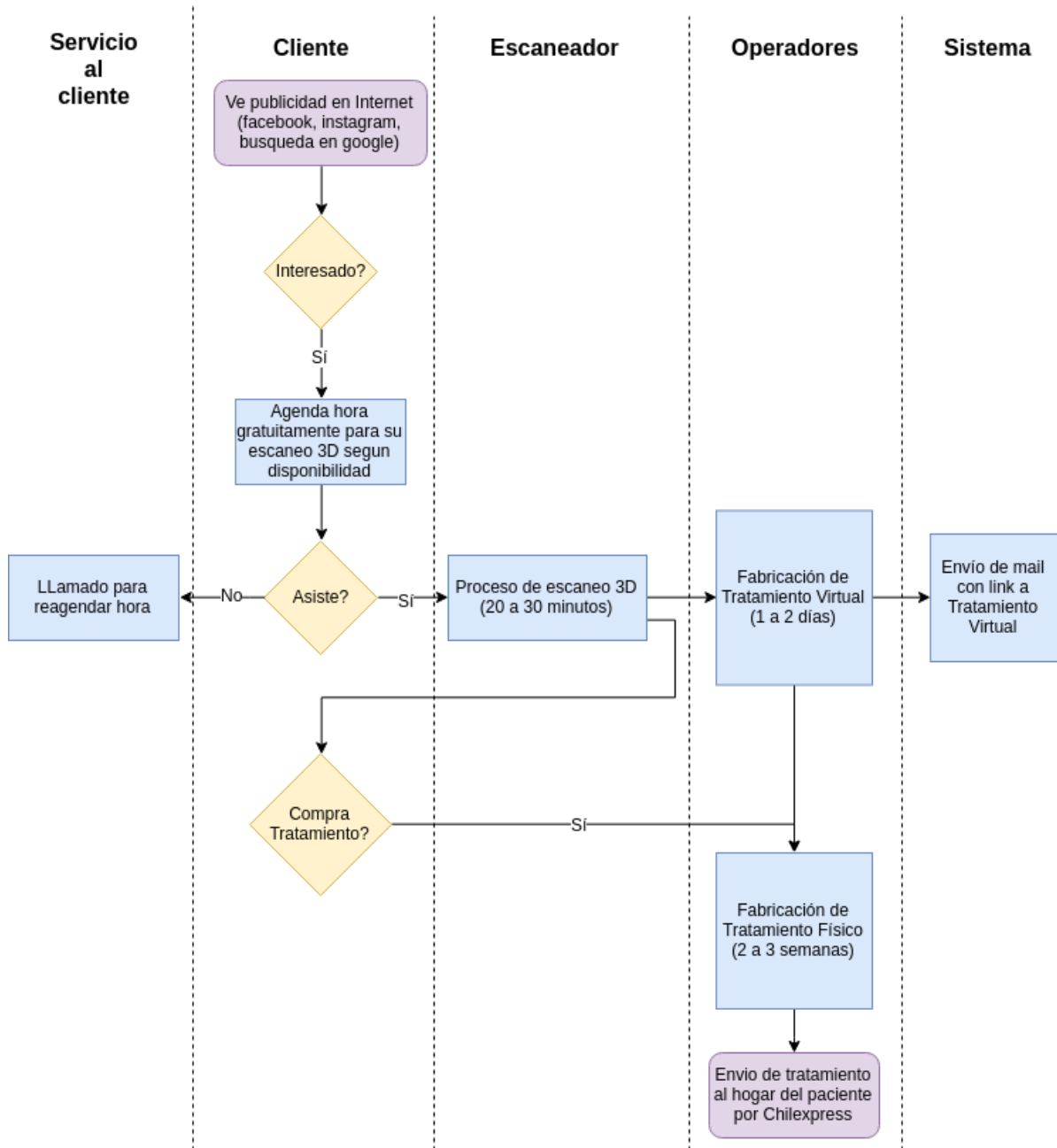


Figura 1.4: Flujo de Negocio Actual

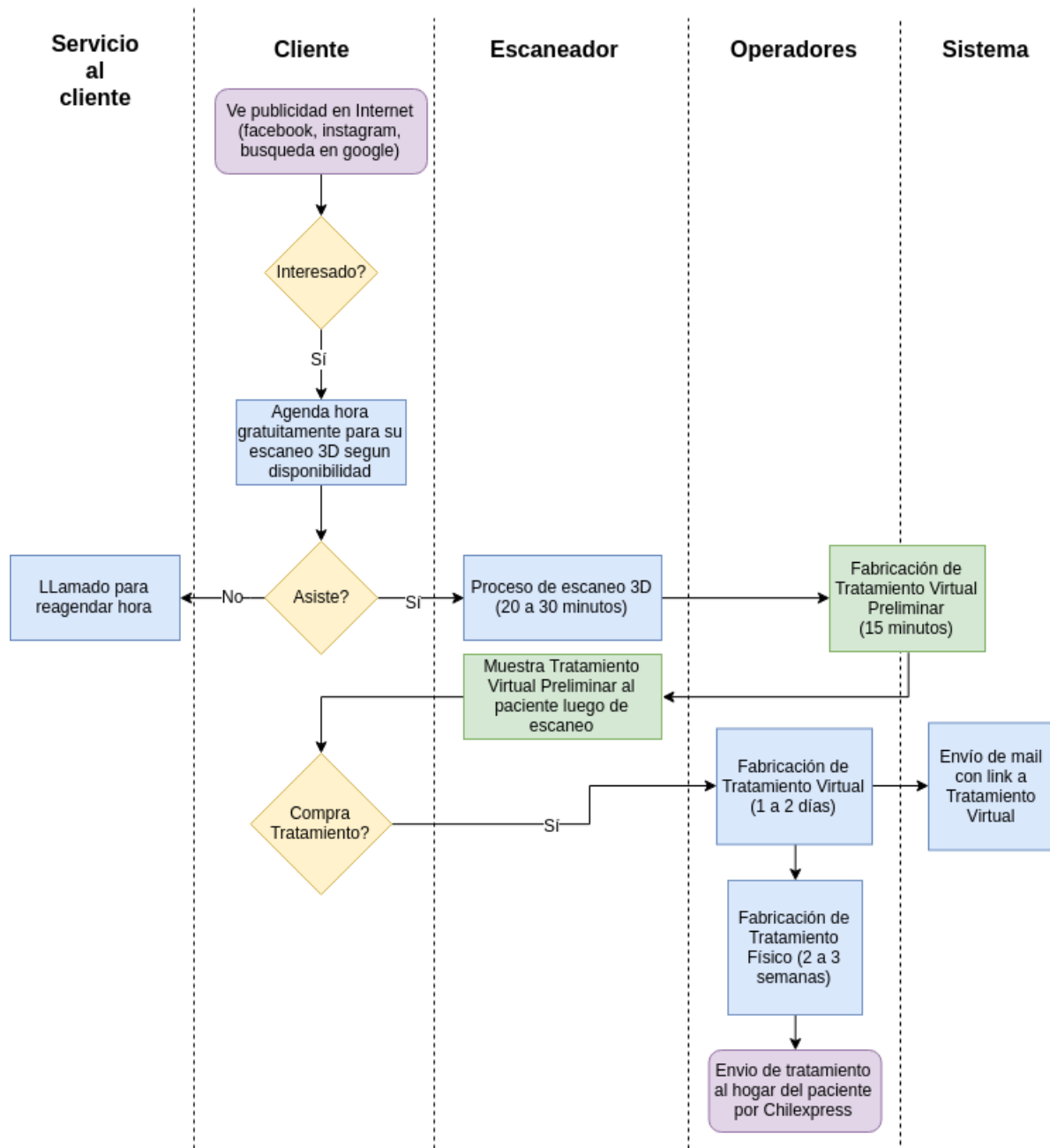


Figura 1.5: Flujo de Negocio Esperado

## 1.2. Objetivos

### 1.2.1. Objetivo general

Diseñar e implementar un sistema que pueda alinear modelos dentales 3D de forma automática, de modo que se pueda visualizar una sonrisa más estética de la que el paciente tiene actualmente, entregando un resultado que sigue los criterios de los tratamientos que se hacen actualmente de forma manual.

### 1.2.2. Objetivos específicos

- El sistema debería ser un módulo dentro de los sistemas actuales de Wizz, de tal forma que su resultado pueda ser visualizado ahí y se de la posibilidad de que este resultado sea refinado manualmente con la misma interfaz que se hace en la actualidad
- El sistema no debería producir resultados que sean estéticamente inferiores al estado inicial del paciente, ya que el sistema en este caso empeoraría la eficiencia de un operador que hace todo de forma manual
- Optimizar lo más posible el sistema de modo que el proceso completo del *Tratamiento Virtual Preliminar* sea en un tiempo esperable por el paciente mientras esté en el centro de Escaneo (máximo 15 minutos)
- El sistema debería estar enfocado también a que sea una ayuda para realizar el *Tratamiento Virtual* definitivo, por tanto no debería solo proponer un resultado más estético, sino también un resultado posible de realizar por medio del tratamiento de alineadores. Esto implica por ejemplo: no mover excesivamente los dientes por fuera de la línea de la encía actual, no deberían haber colisiones entre los dientes, no se puede mover un diente de forma vertical.
- Proponer un sistema de notificaciones para integrar este sistema de tal forma que por una parte el operador sepa cuando debe realizar su trabajo en el *Tratamiento Virtual Preliminar* y por otra parte que el escaneador sepa cuando este trabajo este listo, para así ser mostrado al paciente.

## 1.3. Metodología

La metodología que proponemos para desarrollar la memoria consta de las siguientes etapas:

1. Identificar los criterios que se ocupan actualmente para alinear los dientes de forma manual.
2. Diseñar un Algoritmo que logre efectuar el alineamiento según los criterios anteriormente identificados.
3. Implementación del Algoritmo.
4. Optimización: reducir los tiempos de ejecución.
5. Integración: añadir este algoritmo a los sistemas actuales de Wizz y agregar notificaciones para que tanto operador y escaneador sepan cuando hacer su trabajo.

6. Validación/Evaluación: definir medidas de eficiencia y eficacia de la solución, obtener los resultados y analizarlos

# Capítulo 2

## Conceptos Básicos

En este capítulo se mostrarán los conceptos básicos previos para entender el desarrollo de esta memoria, esto incluye librerías y frameworks específicos, arquitectura y finalidad de los sistemas actuales de Wizz y su estado actual.

### 2.1. Django

Django es un framework web para el lenguaje de programación python, el cual es gratuito y opensource. Una de sus ventajas principales es que cuenta con una comunidad bastante activa y una documentación clara y detallada. [6]. Posee licencia BSD, que permite utilizar, modificar y extender su código fuente de forma gratuita, incluso con fines comerciales. Esto ha permitido que existan extensiones del framework hechas por la comunidad que solucionan las necesidades más usuales dentro del mundo de la programación web.

Django posee un sistema de ORM (Object Relational Mapping). Esto consiste en un conjunto de clases que permiten representar una entidad relacional, en otras palabras una tabla dentro de la base de datos, como si fuera un conjunto de objetos. Las operaciones principales de insertar, actualizar, borrar y seleccionar datos de una tabla se pueden realizar por medio de métodos, abstrayendo la sintaxis SQL. En particular en Django cada clase que hereda de la clase *Model* (provista por Django), esta asociada a una tabla de la base de datos.

Un software desarrollado en Django conforma un *project*, el cual puede contener una o varias *applications*, las cuales permiten mantener una arquitectura modular dentro del proyecto. Cada aplicación usualmente tiene los siguientes archivos principales:

- `models.py`: se definen un conjunto de clases que heredan de la clase de Django *Model*, estas clases definen las tablas dentro de la base de datos y las relaciones entre ellas.
- `views.py`: contiene un conjunto de funciones llamadas views las cuales reciben como argumento un request. Usualmente realizan consultas o modificaciones a la base de datos por medio de la ORM, para esto utilizan las clases y métodos definidos en el archivo `models.py`. Luego retornan un objeto *HttpResponse*, que en la mayoría de los



casos está asociado a un *template* y un diccionario con los datos obtenidos de las queries, el cual Django se encarga de renderizar a un HTML. Este *HttpResponse* también puede contener texto plano o json.

- `url.py`: contiene un conjunto de pares que relacionan una url a su view correspondiente

Este framework sigue el modelo MVT (Model - View - Template) similar al modelo MVC. Como muestra la figura 2.1, inicialmente el usuario desencadena un `Http Request` a través del navegador a una *url* en específico. El url dispatcher de Django ejecuta la *view* correspondiente, función que contiene gran parte de la lógica. Esta función obtiene o modifica datos de los *modelos*. Una vez que obtiene la data la *view* utiliza un *template* para definir la forma en que se mostrarán estos datos al usuario. Django se encarga de renderizar este template a HTML, Javascript y CSS.

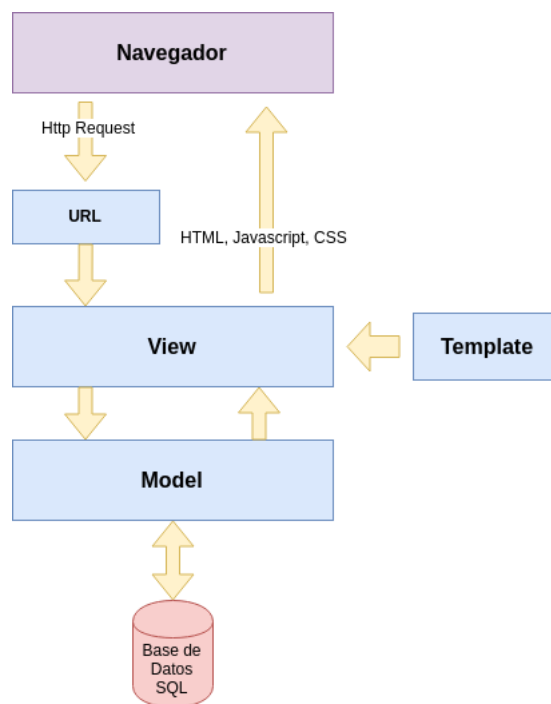


Figura 2.1: Estructura de Django

## 2.2. Three.js

Three.js es una librería liviana escrita en JavaScript para crear y mostrar gráficos animados en 3D en un navegador Web. Su código se encuentra en Github. [10]

Posee el concepto de *Escena* en la cual es posible añadir, eliminar y modificar *objetos 3D* en tiempo de ejecución y así también *cámaras* y *luces*.

En three.js un objeto 3D puede crearse programáticamente, o puede ser importado desde un archivo de texto plano o binario de formatos como *stl* o *ply*, entre otros. Aparte de poder modificar este objeto estructuralmente por ejemplo, modificando la posición de sus vértices, esta librería también cuenta con funciones para trasladar, rotar y escalar los objetos 3D dentro de la escena, además de añadirles textura, color y propiedades de opacidad/brillo. [9]

### 2.2.1. Estructura de un Objeto 3D

Un objeto 3D en threejs es representado por un arreglo de vértices. Un vértice tiene tres componentes, que corresponden a las coordenadas  $x, y, z$ . A su vez cuentan con un arreglo de caras triangulares, las cuales hacen referencia a tres vértices por medio del índice que poseen en el arreglo de vértices, por lo que también una cara se representa como una tupla de tres componentes. Además threejs permite hacer fácilmente operaciones con vectores y matrices en 3 dimensiones.

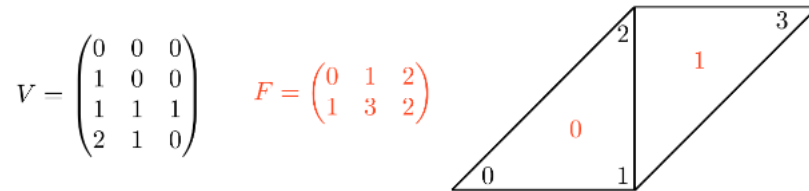


Figura 2.2: Representación de Objeto 3D

La figura 2.2 muestra un ejemplo de objeto 3D de dos caras triangulares.  $V$  corresponde al arreglo de vértices y  $F$  corresponde al arreglo de caras.

### 2.2.2. Sobre los ejes de coordenadas

La posición por defecto de threejs para la cámara, que es a su vez la misma que se utiliza en las aplicaciones web de Wizz, considera el eje  $x$  yendo de izquierda a derecha (horizontal), el eje  $y$  desde abajo hacia arriba (vertical), y el eje  $z$  saliendo perpendicularmente hacia afuera de la pantalla. Así lo muestra el modelo a la izquierda en la figura 2.3. Sin embargo esto puede ser modificado moviendo la cámara, lo que se puede hacer libremente en las aplicaciones web de Wizz. Otra de las vistas más utilizadas es la vista transversal que representa el modelo de la derecha, en donde el eje  $z$  pasa a ponerse en vertical, de abajo hacia arriba, además es visible la zona donde se encuentra el origen  $(0, 0, 0)$ .

La posición de los modelos dentales con respecto a estos ejes es ajustada manualmente de forma aproximada y siempre sigue la orientación mostrada en la figura 2.3.

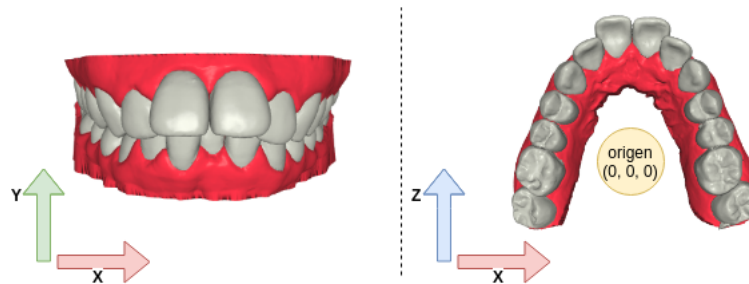


Figura 2.3: Ejes y orientación de los modelos dentales

## 2.3. Arquitectura de los Sistemas de Wizz

Wizz posee cuatro aplicaciones web con diferentes objetivos. Cada una de estas aplicaciones funcionan en servidores distintos y poseen bases de datos distintas. Estos servidores son provistos por *Amazon Web Services*. Cada servidor tiene un costo mensual asociado a la cantidad de uso y capacidad (tipo de CPU, memoria). Las cuatro aplicaciones web están hechas utilizando el framework Django, y utilizan la librería *threejs* en el frontend para visualización y manipulación de modelos tridimensionales.

- Silo (*silowizz.cl*): Corresponde al sistema de información principal, en donde se ingresa, almacena y modifican datos tales como:
  - información de prospectos
  - agendas de horas para escaneos dentales
  - escaneos dentales y sus archivos 3D
  - ventas de tratamientos
  - seguimiento de la etapa en que se encuentra la fabricación de un tratamiento virtual o físico
  - información sobre el envío del tratamiento físico al domicilio del paciente

Este sistema es de uso exclusivo de los funcionarios de Wizz.

- Procesador (*procesadorwizz.cl*): tiene como objetivo principal la *segmentación* de los modelos dentales 3D obtenidos en el escaneo. Es decir, dividir, el archivo 3D inicial obtenido desde el escáner en varios archivos individuales: un archivo en formato ply por cada diente y uno para la encía correspondiente.
- Editor (*editorwizz.cl*): Es la aplicación en donde un operador realiza los movimientos en las piezas dentales previamente segmentadas, para llegar al estado deseado de la sonrisa del paciente.
- Sonríe (*sonriewizz.cl*): Corresponde al sistema donde los pacientes pueden visualizar sus *Tratamientos Virtuales*, y agendar hora para escaneos. Es el único sistema al cual el público general tiene acceso.

Todos estos sistemas tienen distintos grados de comunicación entre sí. Tanto el *Editor* como el *Procesador* son los sistemas clave dentro de la generación de un *Tratamiento Virtual*. En cuanto a los *Tratamientos Virtuales* la aplicación *Silo* solo se dedica a monitorear el estado de avance de la fabricación del tratamiento y contener los archivos iniciales del escaneo, mientras que en el *Procesador* y *Editor* ocurre todo el procesamiento.

El *Procesador* se encarga de guardar los archivos de modelos 3D generados en la *segmentación*. Una vez que la *segmentación* ya ha finalizado el *Editor* puede “preguntar” al *Procesador* por medio de una API cuales son los pacientes disponibles y luego acceder por medio de la misma API a las piezas dentales de un paciente en particular. Una vez que el editor haya cargado todos estos archivos se puede realizar las traslaciones y rotaciones a estas piezas.

Una vez finalizado los movimientos en el *Editor* al haber llegado al estado deseado de la dentadura, las posiciones finales de los dientes se envían de vuelta al *Procesador*. El *Procesador* realiza una interpolación entre el estado inicial y final, dando origen a los modelos 3D que contienen los movimientos intermedios correspondientes a cada alineador.

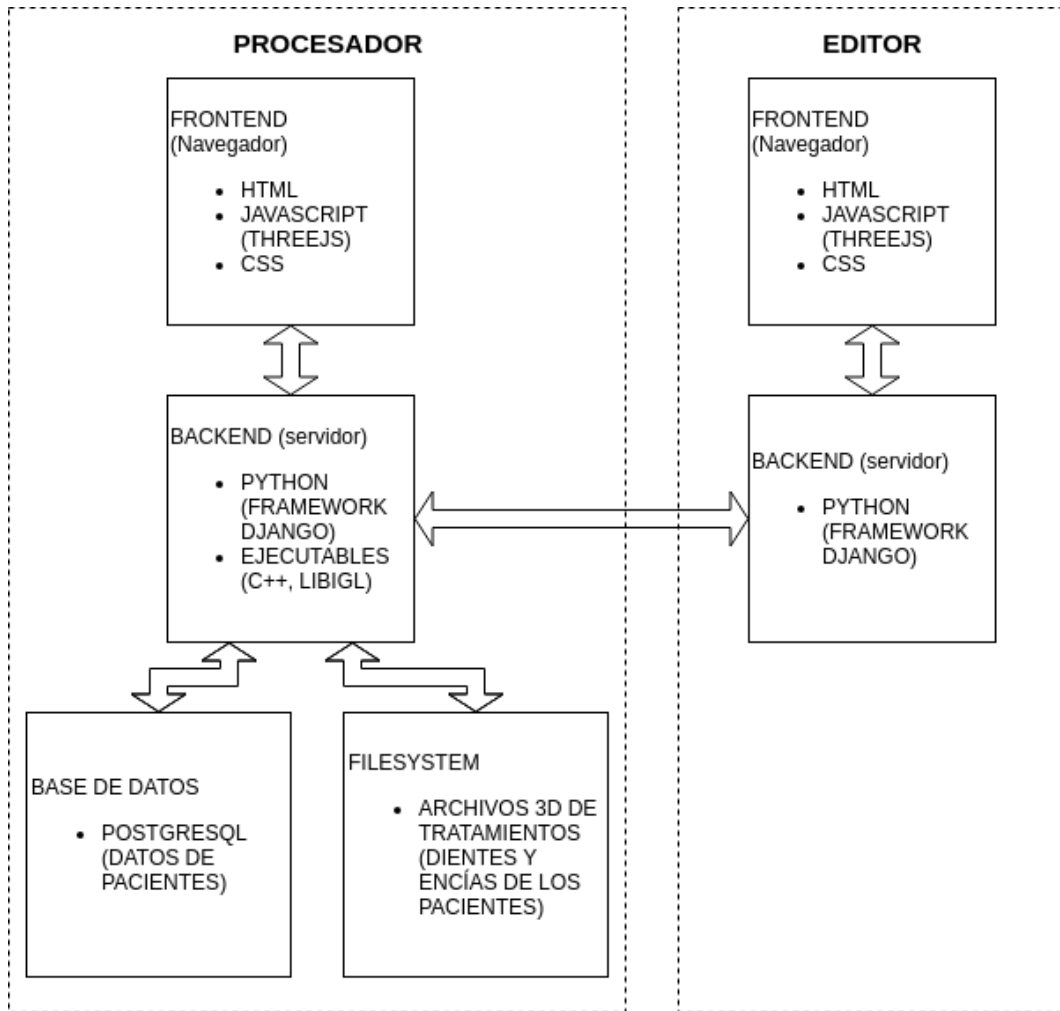


Figura 2.4: Diagrama de los dos sistemas web principales de Wizz

## 2.4. Piezas dentales

Una dentadura humana adulta normal que no haya tenido extracciones considera 28 piezas dentales, 14 en cada arcada. También puede poseer hasta 32 piezas en total y 16 por cada arcada en caso de que a la persona se le hayan desarrollado las muelas del juicio (terceros molares). Esto último depende de la genética de la persona.

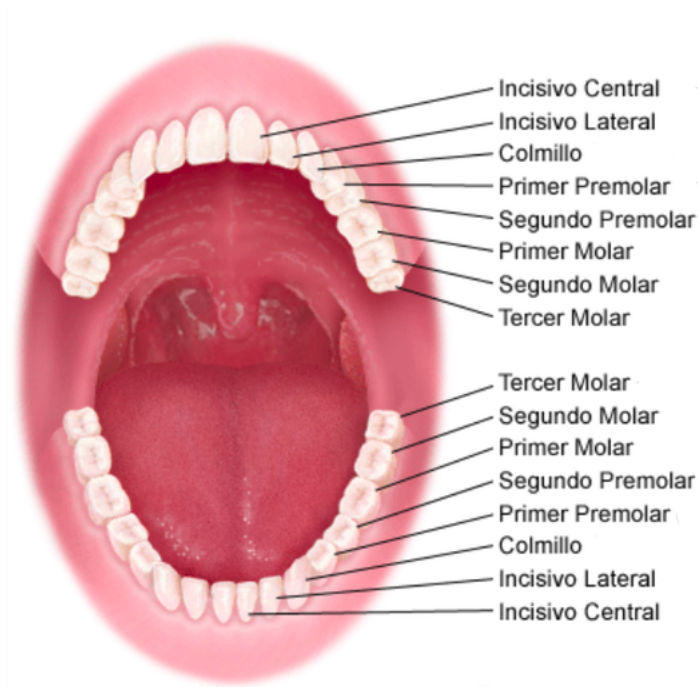


Figura 2.5: Nombres de piezas dentales

La figura 2.5 muestra una de las denominaciones usadas para hacer referencia a los dientes. Otras consideran sistemas numéricos más complejos. Sin embargo la mostrada en la figura es una de las más utilizadas dentro de Wizz. Para hacer referencia a un diente en particular basta con añadir el lado de la pieza (izquierda/derecha) y la arcada en que se encuentra (superior/inferior). Por ejemplo: *incisivo lateral superior derecho*, *primer premolar inferior izquierdo*.

## 2.5. Proceso para crear un Tratamiento Virtual en Wizz

El proceso de creación de un *Tratamiento Virtual* consta de 2 etapas que se describen a continuación:

1. Segmentación
2. Edición (alineamiento)

### 2.5.1. Segmentación

Lo que se obtiene del escaneo de un paciente es un modelo completo de la dentadura, que corresponde a dos archivos, uno para el maxilar (la parte de superior), y otro para la

mandíbula (la parte inferior), ambos en formato STL. Como primer paso se convierten estos archivos a formato PLY y se le reduce la cantidad de caras a un límite de 100000, lo que es suficiente para mantener una buena resolución y no sobrecargar los sistemas. Este proceso es hecho por el sistema de forma automática. Luego ocurren una serie de pasos que necesitan la participación de un operador el cual utiliza la aplicación Web denominada *Procesador*. Estos pasos son:

1. Cortar y Sellar: el operador rota y traslada el modelo para dejarlo centrado, luego define los planos de corte. Con estos planos el sistema corta y sella, de tal forma que el modelo quede sin orificios ni puntas en la parte superior e inferior. Esto se muestra en la figura 2.7
2. Identificación de las cúspides de los dientes: el operador posiciona puntos en las cúspides de los dientes y en la encía, estos puntos están asociados a un índice que representa un diente en particular. Esto se muestra en la figura 2.8
3. Harmonic: una vez que los puntos están identificados el sistema puede segmentar utilizando un algoritmo de campos armónicos, definiendo las líneas que limitan un diente de otro diente y las líneas que limitan los dientes de la encía adyacente. Esto se muestra en la figura 2.9. Si el algoritmo falla, por ejemplo, parte de la encía es identificada como diente, o parte de un diente es identificado como encía entonces hay que volver a seleccionar más puntos en la etapa anterior y volver a ejecutar el algoritmo.

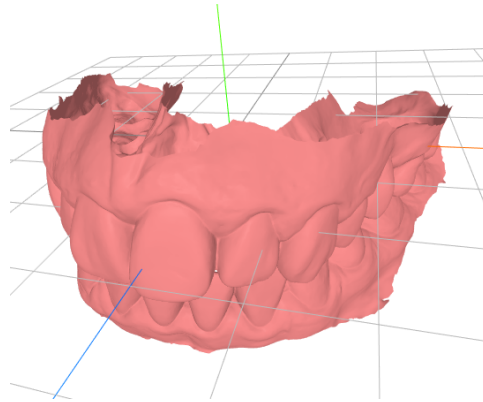


Figura 2.6: Modelo inicial sin modificaciones

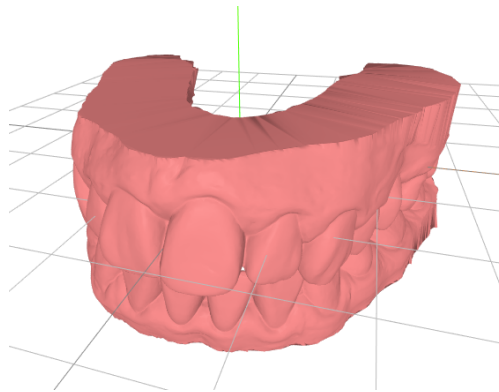


Figura 2.7: Modelo cortado y sellado

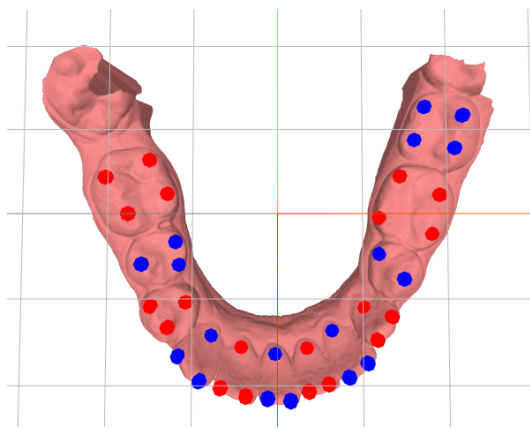


Figura 2.8: Identificación manual de cúspides de los dientes

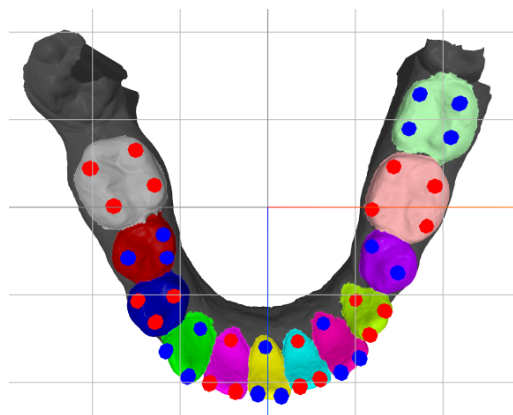


Figura 2.9: Resultado de Harmonic

Finalmente el sistema crea un archivo ply por cada diente y para la encía. El proceso completo de segmentación en general tiene poca incertidumbre y demora un tiempo más o menos constante de 15 minutos.

Los dientes quedan internamente numerados con un índice que se inicia en el cero, y crece de izquierda a derecha. Como existen pacientes con piezas faltantes este índice no indica necesariamente una pieza dental en particular. Sin embargo este índice da información de la posición de un diente con respecto a los demás.

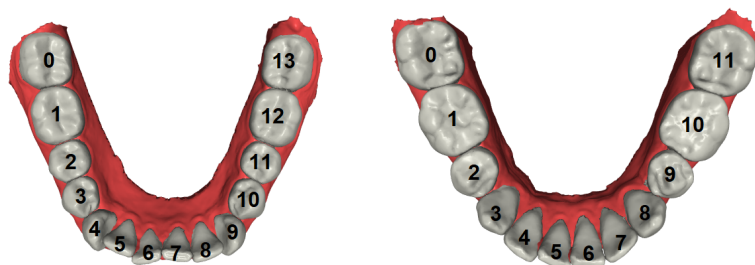


Figura 2.10: Índices de los dientes luego de la segmentación

La figura 2.10 muestra un ejemplo de la numeración interna que asigna el sistema para

un paciente con su dentadura completa (izquierda), y para un paciente con piezas faltantes (derecha).

## 2.5.2. Edición (alineación)

Este proceso es conocido como *Edición* porque el operador lo efectúa en la aplicación web *Editor*. Sin embargo un concepto que describe mejor lo que aquí ocurre sería “alineación”. Esta aplicación web permite realizar los movimientos a cada diente, ya sean traslaciones o rotaciones.

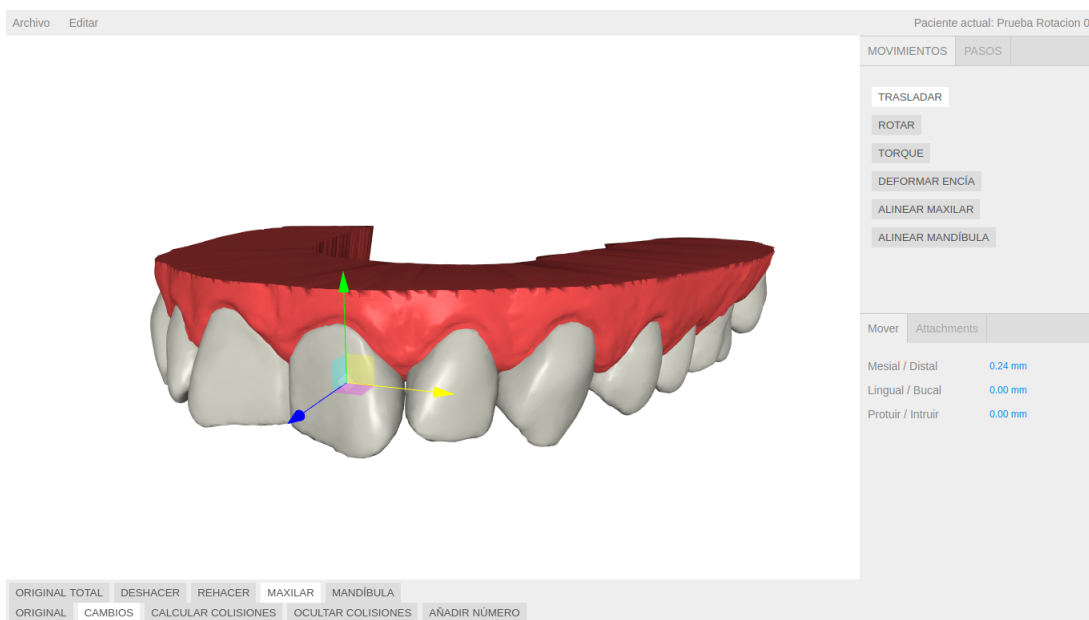


Figura 2.11: Interfaz gráfica del *Editor*

La tarea del técnico es entonces mover cada pieza para llegar al estado final deseado de la dentadura, lo que puede tardar 1 hora o más dependiendo de la gravedad del caso. Principalmente la duración tiene que ver con la dificultad que implica para el operador dejar los dientes sin colisionar (sin que parte del volumen de un diente este dentro de otro diente), ya que los dientes se pueden mover libremente. Es importante dejar este estado final sin colisiones alguna ya que un alineador que haya sido fabricado con un modelo que presenta colisiones estaría tratando de mover los dientes del paciente de una forma que es físicamente imposible.

En el proceso para alinear los dientes manualmente, el técnico trata de intuitivamente formar una curva suave entre las cúspides de los dientes frontales, además de eliminar cualquier inclinación excesiva del diente. Sumado a esto debe trasladar los dientes a través de esta curva imaginaria, de forma que no queden colisiones ni espacios. Cuando hay apiñamiento a veces no es posible dejar los dientes sin colisión, por lo que se debe expandir la arcada, lo que consiste en mover los dientes levemente hacia fuera de la línea de la encía, aumentando el radio de esta curva imaginaria, y en consecuencia dando más espacio para que los dientes puedan posicionarse, de forma contigua mostrando completamente su cara frontal.

Si este movimiento necesita ser muy pronunciado para alcanzar un punto en que no co-



lisionen los dientes, se dice que el paciente no califica, ya que el diente podría quedar con la raíz expuesta si este movimiento es excesivo, ya que este movimiento está limitado por la capacidad que el hueso tiene de adaptarse. Se le sugiere al paciente seguir el tratamiento tradicional, ya que probablemente deba efectuarse una extracción de alguna pieza dental para generar más espacio. De forma análoga, si hay espacios entre los dientes, la arcada se reduce, moviendo los dientes hacia adentro. Si hay demasiado espacio entonces se juntan todos los dientes frontales hacia el centro del punto medio de la línea de la encía, acumulando el espacio sobrante en la zona de las muelas.

Una vez que se llega al estado final se envían las posiciones de los dientes al *Procesador*, y este genera los pasos intermedios. Los pasos intermedios se generan haciendo una interpolación entre el estado inicial de los dientes y el estado final. Esta interpolación esta sujeta a una variable que indica el movimiento máximo que puede haber en un diente entre un paso y el paso siguiente. Además de generar los pasos, el sistema genera la deformación de la encía. Cuando todo esto se guarda, se envía de forma automática a la aplicación web Sonríe, en la cual pueden acceder los pacientes para poder visualizar finalmente su *Tratamiento Virtual*.

## 2.6. Slack

Slack es una herramienta de comunicación para equipos de trabajo que presenta diversos chat. Una de sus características principales es la posibilidad de abrir canales para equipos de trabajo o proyectos en específico. También dentro de los canales se pueden abrir hilos para realizar conversaciones en paralelo. Otras de sus funcionalidades importantes son compartir archivos de cualquier formato (con vista previa si se trata de imágenes fotos, videos, pdf, texto plano), llamadas de audio y video e integración con otros servicios como *github*, *trello*, *google drive*. [7]

Slack ofrece una web API, de tal forma de que es factible provocar ciertas acciones dentro de la aplicación de forma programática desde un sistema ajeno a Slack, como por ejemplo: abrir un canal, cambiar el nombre de un canal, enviar un mensaje a un canal o usuario en específico, invitar un usuario a un canal, enviar recordatorios, etc. [8]

Todos los trabajadores de Wizz poseen una cuenta asociada y deben estar atentos a los mensajes, ya que es el medio de comunicación oficial dentro de la empresa.

## 2.7. Iterative Closest Point

Iterative Closest Point (ICP) es un algoritmo enunciado en 1991 por Yang Chen, Gérard G. Medioni y en 1992 por Paul J. Besl, Neil D. McKay de forma independiente, cuya aplicación principal es registrar la data digitalizada de objetos rígidos que no están fijos en un espacio, con respecto a un modelo geométrico ideal. Registrar en este contexto se refiere a estimar la óptima rotación y traslación que alinea la data 3D obtenida con el modelo geométrico ideal, minimizando la distancia entre ambas formas y permitiendo determinar el grado de equivalencia entre ellas por medio de la métrica de la distancia cuadrática media. Este es un problema clave en el campo de la visión artificial.

El algoritmo puede funcionar con conjuntos de puntos, conjuntos de segmentos de líneas, curvas implícitas, curvas paramétricas, conjuntos de triángulos, superficies implícitas y superficies paramétricas.

El algoritmo procede como sigue: sea el conjunto de puntos  $P$  con un número de puntos  $N_p$  de la data obtenida, y el modelo  $X$  con  $N_x$  primitivas geométricas (puntos, líneas o triángulos). Se aplican los siguientes pasos 1, 2, 3 y 4, iterando hasta alcanzar la convergencia con una tolerancia deseada  $\tau$  [3] [4]

1. calcular los *closest points* entre  $P$  y  $X$  (costo:  $\mathcal{O}(N_p N_x)$  peor caso,  $\mathcal{O}(N_p \log N_x)$  caso promedio)
2. calcular la transformación (traslación y rotación) (costo:  $\mathcal{O}(N_p)$ )
3. aplicar la transformación a  $P$  (costo:  $\mathcal{O}(N_p)$ )
4. terminar de iterar cuando la diferencia en el error cuadrático medio entre los *closest points*, con respecto a la iteración anterior, sea menor a la tolerancia escogida  $\tau$

## 2.8. Ajuste de curvas por medio del método de mínimos cuadrados

### 2.8.1. Ajuste de curvas

La clásica configuración para un problema de ajuste de curvas es un experimento o medición. Las mediciones producen distintos resultados, dependientes de las condiciones experimentales. Típicamente la relación funcional implícita entre las mediciones y las condiciones experimentales es asumida o sabida. Esta relación es descrita por medio de un modelo matemático conocido como función modelo. El objetivo del ajuste de curvas es entonces encontrar los parámetros del modelo que mejor describen la relación entre las mediciones y las condiciones experimentales. [5]

El resultado de una medición es llamado  $y$ , y puede depender de varias condiciones descritas por un vector  $\mathbf{x}$ . Los parámetros del modelo corresponden al vector  $\mathbf{a}$ . En consecuencia la función modelo es:

$$y = f(\mathbf{x}|\mathbf{a})$$

No es posible cambiar las condiciones experimentales de forma continua en la realidad. Se deben seleccionar puntos discretos, indicados por la variable  $i$ , además las observaciones se ven afectadas por errores aleatorios. Por esto el problema se reformula como:

$$y_i = f(\mathbf{x}_i|\mathbf{a}) + \varepsilon_i$$

## 2.8.2. Método de los mínimos cuadrados

El método de los mínimos cuadrados resuelve el problema de encontrar variables desconocidas dado un conjunto de ecuaciones linealmente independientes en el caso sobredeterminado, es decir, cuando se tiene una mayor cantidad de ecuaciones que variables desconocidas.

Como las observaciones  $y_i$  se ven afectadas por los errores expresados por  $\varepsilon_i$  es beneficioso tener la mayor cantidad de ecuaciones posible (es decir, pares de  $y_i$  e  $\mathbf{x}_i$ ) para obtener estadísticas mejores y confiables. Esto significa que los parámetros pueden ser estimados de forma más precisa.

Suponiendo que se tiene un conjunto de observaciones  $y_i$  correspondiente a las condiciones  $\mathbf{x}_i$ , bajo las cuales se hicieron las observaciones. El objetivo de el ajuste de curvas por medio de mínimos cuadrados es minimizar el error residual entre las observaciones y los valores calculados por la función modelo

$$\chi^2 = \sum_i w_i \cdot [f(\mathbf{x}_i|\mathbf{a}) - y_i]^2 \longrightarrow Min$$

donde  $w_i$  es un peso que indica la fiabilidad del dato  $y_i$  obtenido. El problema de ajuste corresponde a un problema de minimización. El valor de  $\chi^2$  es dependiente de una buena elección de la función modelo. [5]

# Capítulo 3

## Algoritmo Propuesto

Este algoritmo corresponde a una automatización del proceso de *Edición* o alineación, especificado en el capítulo de Conceptos Básicos. Se asume que la etapa de *Segmentación* se seguirá haciendo de forma manual, ya que presenta poca complejidad y su duración es más o menos constante. Como este tratamiento de tipo preliminar no finalizará en la impresión, se disminuyó la resolución de los modelos. Actualmente el número de caras que se utiliza en los tratamientos que son impresos es de 100000. Se probó entonces bajar la resolución a 50000 caras, o sea la mitad. Sin embargo esta baja en la resolución reduce la concavidad de la zona de contacto entre el diente y la encía, resultando en errores en el proceso de segmentación.

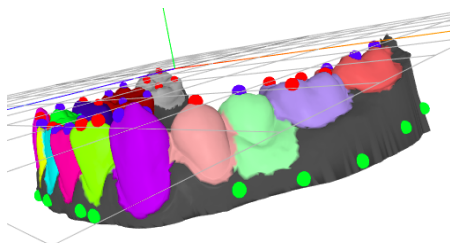


Figura 3.1: Error al intentar segmentar un modelo de baja resolución (50000 caras), parte de la encía es identificada como diente

Se concluyó que el mínimo número de caras para que estos errores no ocurran se encuentra al bajar la resolución a  $2/3$  de la original, es decir 66666 caras. Esto aumenta mucho la performance debido a que existen algoritmos cuadráticos con respecto al número de vértices que se ejecutan dentro de la *Segmentación*. El proceso de segmentación en este contexto será entonces por debajo de los 10 minutos.

### 3.1. Idea general

El algoritmo presentado a continuación está basado en la idea de que las caras frontales de los dientes en una dentadura correctamente alineada forman una curva suave. Esto se puede ver intuitivamente en los tratamientos de ortodoncia por medio de brackets, en la que los brackets están posicionados en medio de la cara frontal del diente. Por este bracket pasa un

alambre, el cual posee la forma de la curva deseada. Si los dientes no están bien posicionados, el bracket experimentará una fuerza por parte del alambre y en consecuencia también el diente que esta pegado al bracket.

El algoritmo consta de las siguientes etapas

- Obtención de una Curva: una curva simétrica que sea similar a la curva natural que el paciente tiene actualmente
- Posición Neutra del diente: en esta etapa podremos establecer la posición inicial necesaria del diente para luego rotarlo el ángulo justo de modo que quede tangencial a la curva previamente calculada
- Posicionamiento de los Dientes en la Curva: en esta etapa se ubica un diente uno al lado del otro a lo largo de la curva de manera tangencial a ella, de modo de no dejar colisiones ni espacios entre los dientes

## 3.2. Sobre la implementación del algoritmo

La mayor parte de la siguiente solución se programó en el lenguaje javascript, como un módulo dentro del frontend del *Editor*. De esta forma se usan todas las funcionalidades para operar objetos 3D que ofrece la librería *threejs* (rotación, traslación, manipulación de vectores, etc). Se añadió código python en el backend del editor en los casos que fue necesario, por ejemplo para leer y guardar archivos en el servidor, y para hacer uso de librerías de python específicas.

## 3.3. Control de versiones

Como sistema de control de versiones se utilizó Git, aprovechando que los software de Wizz ya cuentan con repositorios Git en el servicio Bitbucket. En particular, para mantener el código, pruebas y cambios que en este capítulo se describen se abrió una nueva rama en el repositorio del *Editor*, de forma de tener independencia del código que estaba en ese momento funcionando en producción.

## 3.4. Dientes a alinear por el algoritmo

Muchos pacientes presentan piezas faltantes en los molares, o pueden tener coronas o implantes, por lo que estos no se pueden mover en el tratamiento. Tener piezas faltantes entre los premolares es algo más inusual. Si consideramos además que el tratamiento esta enfocado a la parte estética, los dientes entre los premolares son los que más se notan en la sonrisa de una persona.

El sistema considerará entonces solo ocho dientes por cada arcada, los cuales son los que se encuentran entre los primeros premolares, incluyéndolos. Se asume que el sistema deberá recibir como input el primer premolar del lado izquierdo para cada arcada.

Se considera que identificar el primer premolar izquierdo es una tarea fácil para el operador del software y que además conlleva muy poco tiempo (menos de 5 segundos). Por otro lado

desarrollar un sistema que posea un módulo que pueda identificar cada diente tiene un costo muy alto en cuanto a tiempo de desarrollo (meses). Y el tiempo en que se reducirá el proceso es pequeño, pero es algo que sería necesario solo en caso de que se este buscando la autonomía total del sistema, eliminando completamente la participación humana.

Una vez identificado el primer premolar izquierdo el algoritmo podrá obtener los siete dientes siguientes a base de su índice, sabiendo con certeza de que piezas dentales específicas se tratan. Para esto se asume que el paciente no tiene piezas faltantes entre los primeros premolares, lo cual es cierto en la mayoría de los casos.

Las personas que tengan piezas faltantes entre los premolares se considerarán casos anómalos, y no son casos que el sistema pueda detectar por sí mismo. De igual manera los casos de pacientes con dientes extra ocurren rara vez, y corresponden a dientes de leche que nunca cayeron. Una búsqueda por texto de las observaciones anotadas por los escaneadores en el sistema muestra que han habido 20 ocurrencias dentro de 6105 escaneos. Tanto el caso de pieza faltante, como el de pieza extra, son fácilmente detectables por el operador y en consecuencia podrá decidir no utilizar el sistema, o bien utilizarlo sabiendo que los resultados podrían no tener la calidad esperada.

Durante las diferentes etapas del algoritmo se utiliza una nueva numeración para estos 8 dientes seleccionados, que va desde el 0 al 7. La diferencia con la numeración original obtenida en la segmentación es que en este caso se sabe de que pieza dental específica se trata por medio de la numeración: la 0 corresponderá al primer premolar izquierdo, la 1 corresponderá al canino izquierdo, la 2 corresponderá al incisivo lateral izquierdo, etc.

La figura 3.2 muestra en color púrpura los dientes seleccionados por el algoritmo para ser alineados, estos dientes para efectos internos del algoritmo están numerados del 0 al 7, de izquierda a derecha.

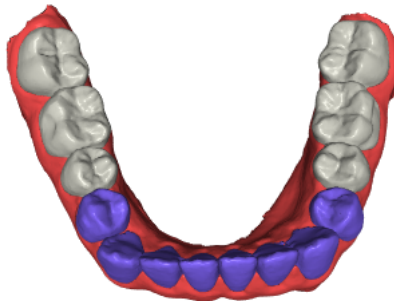


Figura 3.2: Dientes seleccionados para ser alineados

## 3.5. Obtención de una Curva

### 3.5.1. Raycast

El Raycasting es una funcionalidad de threejs que representa un rayo, al cual se le asigna un origen y un vector de dirección. Luego se le entrega un objeto o una lista de objetos 3D, con las que este rayo podría chocar potencialmente. El resultado es una lista de todos los puntos en donde el rayo choca con estos objetos, ordenados del más cercano al más lejano.

La figura 3.3 muestra un ejemplo de uso de raycast, en donde *origin* corresponde a un punto en el espacio 3D, *directionVector* corresponde a un vector y *hitresult* corresponde en consecuencia a un arreglo con los puntos de colisión entre *rcaster* y el objeto 3D *tooth*.

```
var rcaster = new THREE.Raycaster();  
rcaster.set(origin, directionVector.normalize());  
var hitResult = rcaster.intersectObject(tooth);
```

Figura 3.3: ejemplo raycaster

### 3.5.2. Puntos seleccionados para efectuar el ajuste a una curva

Se aplicó un procedimiento para obtener puntos que pasaran por las caras frontales de los dientes pasando por la zona media de estos. Para esto se utilizó la funcionalidad de raycast explicada anteriormente.

La funcionalidad de raycast necesita de un punto de origen y un vector de dirección. Como punto de origen se seleccionó un punto central, calculado como el promedio del centroide (media de todos los vértices del diente) de los segundos premolares. Por su parte, el vector de dirección va cambiando en cada iteración generando un arco. Esto se logró haciendo una interpolación entre los vectores de dirección que se forman al hacer la resta vectorial entre los centroides de los dientes y el punto de origen seleccionado anteriormente. Se obtuvieron 20 vectores de dirección interpolados entre cada para de dientes contiguos.

Al aplicar el raycast utilizando estos vectores de dirección se seleccionaron los puntos de colisión más lejanos, es decir los de la cara exterior del diente.

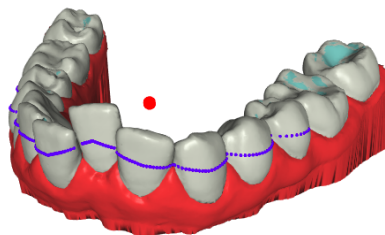


Figura 3.4:

La figura 3.4 muestra en rojo el punto de origen del raycast, y en azul los puntos de colisión más lejanos, que son los seleccionados para realizar el ajuste.

```

var teeth = [];
// obtener vectores de dirección que van desde origen (centro de los premolares) al centro de los dientes
var directionVectors = [];
for (var i = initToothIndex - 1; i <= finalToothIndex + 1; i++) {
    var tooth = getTeethByNumber(jaw, i);
    teeth.push(tooth);
    var directionVector = new THREE.Vector3().subVectors(tooth.position, origin).normalize();
    directionVectors.push(directionVector);
}

var interpolationSteps = 20;
for (i = 0; i < finalToothIndex - initToothIndex + 2; i++) {
    var collisionsPerTooth = [];
    for (var j = 0; j < interpolationSteps; j++) {
        // vector de dirección interpolado
        var directionVectorInterpolated = new THREE.Vector3(
            directionVectors[i].x * (interpolationSteps - j) / interpolationSteps
            + directionVectors[i + 1].x * j / interpolationSteps,
            directionVectors[i].y * (interpolationSteps - j) / interpolationSteps
            + directionVectors[i + 1].y * j / interpolationSteps,
            directionVectors[i].z * (interpolationSteps - j) / interpolationSteps
            + directionVectors[i + 1].z * j / interpolationSteps
        );
        // disparar raycast desde origen con la dirección interpolada
        var rcaster = new THREE.Raycaster(origin, directionVectorInterpolated.normalize());
        // obtener las colisiones con el conjunto de dientes
        var collisions = rcaster.intersectObjects(teeth);
        // si hay colisión
        if (collisions.length > 0) {
            // elegir la colisión mas lejana
            var farestCollision = collisions[collisions.length - 1];
            collisionsPerTooth.push({point: farestCollision.point, distance: farestCollision.distance})
        }
    }
    farestCollision.sort(compareFunction).reverse();
    for (j = 0; j < collisionsPerTooth.length * 0.5; j++) {
        xy.push([collisionsPerTooth[j].point.x, collisionsPerTooth[j].point.y]);
        xz.push({x: collisionsPerTooth[j].point.x, z: collisionsPerTooth[j].point.z});
    }
}

```

Figura 3.5: Código para obtención de los puntos seleccionados

### 3.5.3. Deducción de una ecuación

Luego de revisar la literatura se encontró que la siguiente función ajusta bien la forma de un arco dental bien alineado, y es capaz de expresar los arcos de forma ovoide, triangular y cuadrado [2]

$$f(x) = ax^6 + bx^2$$

Como los modelos utilizados en los tratamientos no siempre están perfectamente centrados se agregaron las constantes  $c$  y  $d$ , que representan el posible desplazamiento horizontal y vertical del modelo:

$$f(x) = a(x - c)^6 + b(x - c)^2 + d$$

Para ajustar esta curva a los puntos seleccionados se utilizó la función `curve_fit` de la librería de python `scipy` la cual utiliza la técnica de los mínimos cuadrados para ajustar una curva a un conjunto de puntos que provee el usuario, dando como resultado los parámetros  $a$ ,  $b$ ,  $c$ ,  $d$  de la ecuación.



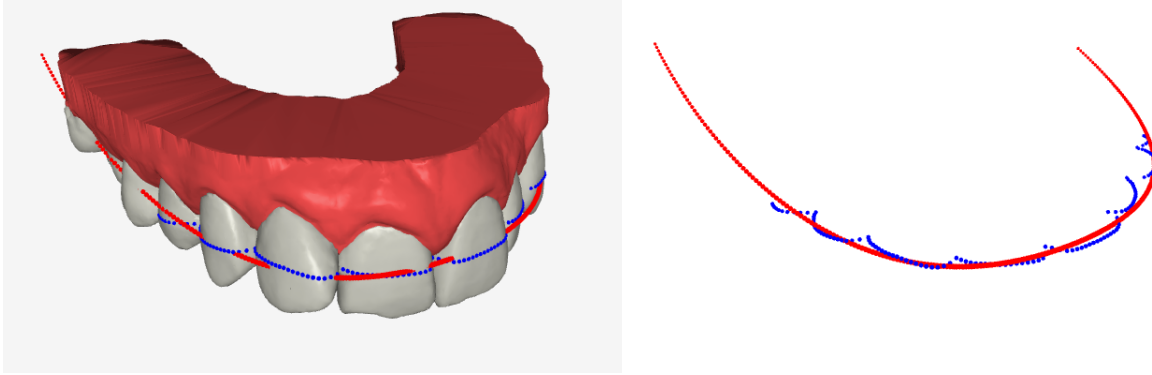


Figura 3.6: Los puntos azules corresponden a los puntos seleccionados para ajustar la curva, y los puntos rojos corresponden a puntos de la curva resultante

```
def jaw_function(x, a, b, c, d):
    return a * (x - c) ** 6 + b * (x - c) ** 2 + d

@login_required(login_url='/user/login/')
def constrained_jaw_regression(request):
    if request.method == "POST":
        data = json.loads(request.POST['data'])
        x_data = []
        z_data = []
        for point in data:
            x_data.append(point['x'])
            z_data.append(point['z'])
        x_np_array = np.array(x_data)
        z_np_array = np.array(z_data)
        coefs, _ = curve_fit(jaw_function, x_np_array, z_np_array)
        response_data = {'a': coefs[0], 'b': coefs[1], 'c': coefs[2], 'd': coefs[3]}
        return HttpResponse(json.dumps(response_data), content_type="application/json")
    else:
        return HttpResponse(status=500)
```

Figura 3.7: `jaw_function` define la forma de la función a la cual se ajustará la curva, y `constrained_jaw_regression` lee los puntos seleccionados y aplica `curve_fit`

```
function JawCurve(a, b, c, d) {
    this.a = a;
    this.b = b;
    this.c = c;
    this.d = d;
}

JawCurve.prototype.evaluate = function (x) {
    return this.a * Math.pow(x - this.c, 6) + this.b * Math.pow(x - this.c, 2) + this.d;
};
```

Figura 3.8: Luego de que se obtiene el resultado, este se utiliza para crear un objeto javascript de tipo `JawCurve`, el cual tendrá métodos asociados, por ejemplo `evaluate`.

## 3.6. Posición Neutra del diente

La posición Neutra del diente se refiere a hacer calzar los ejes del diente con los ejes  $x$ ,  $y$ ,  $z$  globales, de tal manera que el eje del diente que sigue la línea de la encía, o sea el eje mesial-distal, coincida con el eje  $x$ , el eje bucal-lingual con el eje  $z$ , y intrusión-protusión con el eje  $y$ . Dejar los dientes en esta posición es necesario para luego rotar este diente justo de forma que su cara exterior quede tangente a la curva calculada previamente. La cantidad de radianes a rotar se puede obtener luego aplicando el arcotangente de la derivada de la curva.

### 3.6.1. Obtención manual de dientes en posición neutra

Para lograr esta posición, se tomaron los dientes de una persona en particular y se alinearon manualmente a los ejes globales dentro del *Editor*. Se implementó un código de tal forma que por medio de la consola de Chrome, se llama a una función de javascript. Esta función recibe el objeto 3D del diente alineado manualmente, y luego llama a una view de Django en el backend del *Editor* la cual guarda la nube de puntos del diente en su posición neutra en un archivo en el servidor. Se hizo este proceso para todos los dientes a alinear, o sea los ocho dientes entre los premolares para el maxilar y los 8 dientes entre los premolares para la mandíbula. El nombre de el archivo indica a cual diente corresponde por medio de un índice y a que arcada pertenece, además de un identificador del paciente.

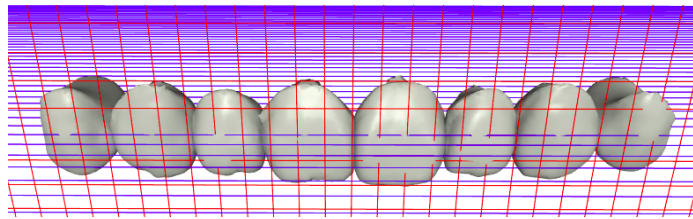


Figura 3.9: Proceso para ubicar manualmente dientes en posición neutra

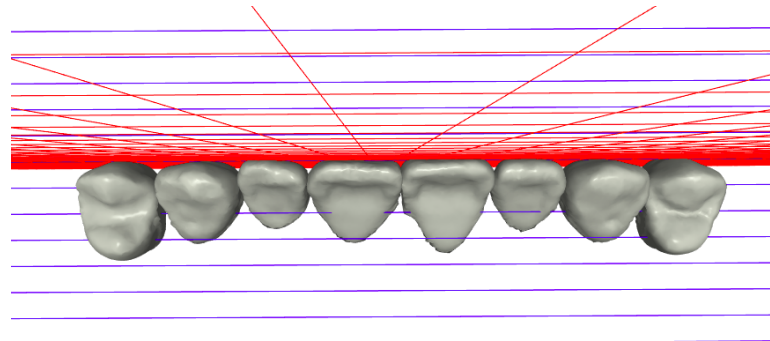


Figura 3.10: Proceso para ubicar manualmente dientes en posición neutra



Figura 3.11: Directorio en el servidor con archivos de nubes de puntos correspondientes a dientes en su posición neutra. Muestra la arcada, índice del diente e ID del paciente del cual se obtuvieron

### 3.6.2. Aplicación de ICP

Luego para obtener un diente en su posición neutra se puede aplicar el algoritmo de Iterative Closest Point. Este algoritmo minimiza la diferencia entre dos nubes de puntos. El algoritmo entrega como output la matriz de transformación, la cual aplicada a la nube de puntos a modificar, minimiza la diferencia entre ambas nubes de puntos. [3]

Se implementó una función javascript que recibe el objeto 3d de un diente y luego envía la nube de puntos del diente a una view de Django en el backend. Esta view busca el archivo correspondiente al diente con igual índice y misma arcada en su posición neutra, y aplica ICP entre ambas nubes de puntos. Luego envía la matriz de transformación de vuelta al frontend, y se le aplica la transformación al diente torcido, quedando en una posición neutra.

Se utilizó una implementación de ICP en python obtenida de: <https://github.com/ClayFlannigan/icp>

El resultado no es 100% exacto debido a que las nubes de puntos no corresponden a objetos exactamente iguales debido a que los dientes corresponden a pacientes distintos.

Se propone como solución entonces aumentar los modelos de dientes en posición neutra guardados en el servidor, de tal forma de aumentar las probabilidades de que al aplicar la operación a un diente aleatorio, este pueda compararse con una nube de puntos que sea más similar a este diente. Una forma de saber esto es elegir la matriz de transformación cuya ejecución de ICP retorne el error más pequeño. El error corresponde al promedio de la distancia entre los puntos que el algoritmo trató de hacer calzar entre ambas nubes.

La figura 3.12 muestra como actúa el algoritmo de ICP. Podemos ver como la cara exterior de los dientes queda alineada con el plano  $x - y$ , para los 8 dientes a alinear.

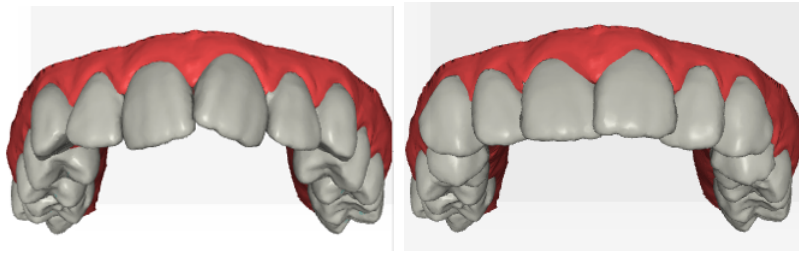


Figura 3.12: La imagen de la izquierda muestra la dentadura original de un paciente. La de la derecha muestra la misma dentadura luego de haber aplicado ICP a los dientes

## 3.7. Posicionamiento de los Dientes en la Curva

El posicionamiento en la curva ocurre una vez que se tiene la formula de esta y también se obtuvo la posición neutra de los dientes.

En el posicionamiento se espera que los dientes queden con su cara exterior tocando de forma tangencial a la curva calculada, mientras que no debe haber colisiones y espacios entre los dientes. Para lograr esto se tuvo que modificar el algoritmo de colisión.

### 3.7.1. Modificación del Algoritmo de Colisión

El *Editor* contenía anteriormente un algoritmo de calculo de colisiones, el cual entregaba gráficamente un objeto 3D que representa la intersección entre dos dientes. Este objeto 3D solo sirve como referencia para la persona que ocupa el editor manualmente y no entrega ninguna información sobre la magnitud de la colisión.

El algoritmo de colisión se modificó para que entregara una variable cuantitativa de la magnitud de la colisión. Esta variable representa la profundidad aproximada de la colisión en milímetros. Además en caso de que esta variable sea negativa, esta representa la separación entre ambos dientes.

El algoritmo funciona con un raycast con origen en el centro de uno de los dientes, orientando su dirección a uno de los vértices del mismo diente. Luego se verifica la colisión de este raycast con el otro diente. Esto se realiza para todos los vértices. `maxCollisionWidth` representa entonces la distancia máxima entre dos puntos que se encuentran en la superficie de dientes distintos, en el caso en que ambos puntos están dentro del volumen del otro diente. Se acepta un error absoluto de 0.1 milímetros.

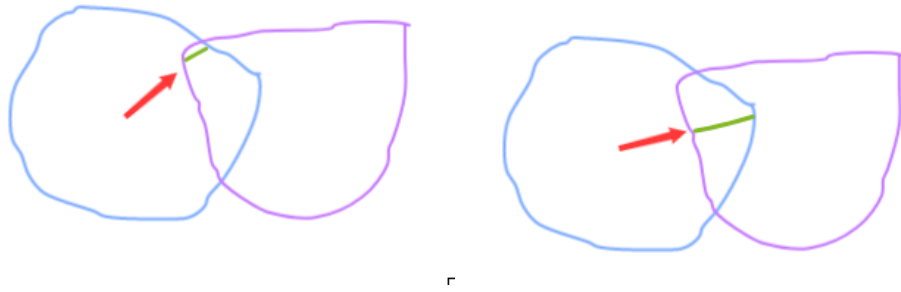


Figura 3.13: Representación del algoritmo, la flecha roja representa el raycast que sale desde el centro del diente, y el segmento verde corresponde a collisionWidth. El segmento verde en el caso de la derecha correspondería a maxCollisionWidth

```

function checkCollisionNumber(obj1, obj2) {
    var maxColllisionWidth = -Infinity;
    var rcaster = new THREE.Raycaster();
    obj1.updateMatrixWorld();
    obj2.updateMatrixWorld();
    // para todos los vertices del obj1
    for (var vi = 0; vi < obj1.geometry.vertices.length; vi++) {
        // convertir vertice de obj1 a cordeenada global
        var global_vertex = obj1.geometry.vertices[vi].clone().applyMatrix4(obj1.matrix);
        // vector de direccion del vertice segun el origen de obj1
        var dirv = global_vertex.sub(obj1.position);
        // setup raycaster, desde el centro de obj1 hasta el vertice vi de obj1
        rcaster.set(obj1.position, dirv.clone().normalize());
        // obtener el resultado de la colisión de rcaster con obj2
        var hitResult = rcaster.intersectObject(obj2);
        if (hitResult.length > 0) { //si es que existió colisión con obj2
            var collisionWidth = dirv.length() - hitResult[0].distance;
            if (collisionWidth > maxColllisionWidth) {
                maxColllisionWidth = collisionWidth;
            }
        }
    }
    // igual a lo anterior pero desde obj2
    for (vi = 0; vi < obj2.geometry.vertices.length; vi++) {...}

    if (maxColllisionWidth > 0 && maxColllisionWidth <= 0.1) {
        return 0;
    }
    else if (maxColllisionWidth < 0 && maxColllisionWidth >= -0.1){
        return 0
    }
    else {
        return maxColllisionWidth
    }
}

```

Figura 3.14: Código del algoritmo de colisión

### 3.7.2. Cálculo de las medidas límites del Diente

Cuando el diente está en su posición neutra se calcularon las medidas de ciertos segmentos utilizando raycast. Estas medidas ayudan a posicionar los dientes aproximadamente de forma rápida, para luego refinar el posicionamiento usando el algoritmo de colisión anteriormente descrito. De esta forma no es necesario iterar tantas veces sobre el algoritmo de colisión, el cual es costoso.

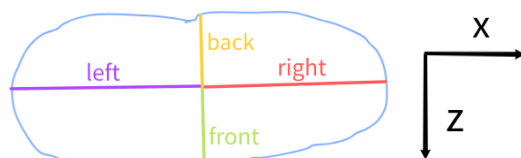


Figura 3.15: Vista del plano XZ con  $Y=0$ , y los 4 medidas límites calculadas

### 3.7.3. Posicionamiento

Los dientes se posicionan uno a uno de izquierda a derecha. El primer diente se ubica en el punto de la curva más cercano a su posición original. Luego se le aplica la rotación para que su ángulo sea tangencial a la curva. Posteriormente se mueve en la medida límite *front* en dirección perpendicular a la tangente, de esta forma la cara exterior queda justo tocando la curva.

```

JawCurve.prototype.evaluate = function (x) {
    return this.a * Math.pow(x - this.c, 6) + this.b * Math.pow(x - this.c, 2) + this.d;
};

JawCurve.prototype.derivative = function (x) {
    return 6 * this.a * Math.pow(x - this.c, 5) + 2 * this.b * (x - this.c);
};

JawCurve.prototype.tangentAngle = function (x) {
    return Math.atan(this.derivative(x));
};

```

Figura 3.16: métodos para calcular la derivada y al ángulo de la tangente en cierto punto de la curva

Luego para ubicar el diente siguiente se avanza a través de la curva desde la posición del diente anteriormente ubicado una distancia de  $right(diente\_anterior) + left(diente\_actual)$ . El diente se ubica ahí y se mueve una distancia *front* en dirección perpendicular a la tangente, igual que en el caso anterior. Sin embargo esto no asegura que no haya colisión. Se calculan colisiones entre ambos, y si es distinta a cero, se avanza el punto de posición del diente a través de la curva una longitud igual a lo entregado por el algoritmo de colisión (`maxCollisionWidth`). Se itera hasta que esto converja y la colisión entre ambos sea cero (estrictamente la colisión no es cero, sino menor a 0.1, debido al error que acepta el algoritmo de colisión). En general la colisión entre ambos llega a cero en menos de 4 iteraciones. En el caso de separaciones el procedimiento es análogo ya que lo único que cambia es el signo del resultado del algoritmo de colisión.

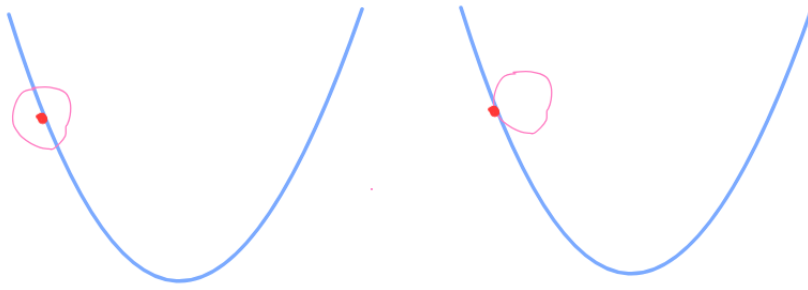


Figura 3.17: Se mueve el diente en la medida límite *front* en dirección perpendicular a la tangente

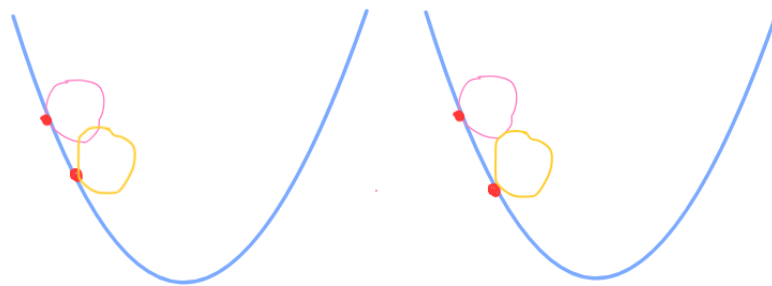


Figura 3.18: En el primer posicionamiento (izquierda), la distancia entre ambos puntos rojos a través de la curva es de  $right(diente\_anterior) + left(diente\_actual)$ . Luego se calcula la colisión entre ambos dientes, y se avanza el segundo punto rojo en el resultado entregado por el algoritmo de colisión de forma iterativa hasta llegar a cero.

Se sigue el mismo procedimiento hasta completarlo con los 8 dientes a alinear. El procedimiento anterior asegura que los 8 dientes no contengan espacios o colisiones entre sí. Sin embargo el último diente ubicado puede quedar colisionando o con espacios respecto del diente siguiente. Esto se resuelve escalando la curva y repitiendo el proceso de posicionamiento.

### 3.7.4. Escalamiento de la curva

Cuando el último diente ubicado que está considerado por el algoritmo, queda colisionando o con espacios con el diente siguiente, la curva se escala. El porcentaje a escalar corresponde al espacio sobrante o faltante dividido en la longitud completa del arco utilizado para los 8 dientes. Luego de escalar se posicionan los dientes nuevamente.

Si después de esto quedan nuevamente espacios o colisiones, el procedimiento de escalamiento se aplica de nuevo, pero sobre el escalamiento anterior y así sucesivamente hasta converger a cero colisión o espacio.

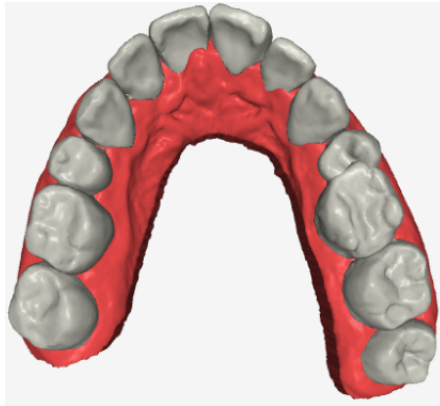


Figura 3.19: Caso en que el último premolar ubicado queda colisionando con el molar siguiente

```

JawCurve.prototype.scale = function (factor) {
  this.a = factor * Math.pow(1 / factor, 6) * this.a;
  this.b = factor * Math.pow(1 / factor, 2) * this.b;
  this.d = factor * this.d;
};

```

Figura 3.20: método para escalar la curva

```

if (teethCounter === 8) {
  tooth = getTeethByNumber(jaw, leftPremolarIndex + teethCounter);
  previousTooth = getTeethByNumber(jaw, leftPremolarIndex + teethCounter - 1);
  distance = checkCollisionNumber(previousTooth, tooth);
  var totalLength = jawCurve.arcLength(initX, finalX);
  if (distance === 0) {
    break;
  }
  else if (distance > 0) {
    jawCurve.scale(1 + Math.abs(distance / totalLength));
    console.log("se escalo en" + Math.abs(distance / totalLength));
  }
  else {
    jawCurve.scale(1 - Math.abs(distance / totalLength));
    console.log("se escalo en" + (-Math.abs(distance / totalLength)));
  }
  teethCounter = 0 // el posicionamiento parte de nuevo
}

```

Figura 3.21: casos en que es necesario escalar



## 3.8. Optimizaciones

### 3.8.1. ICP en paralelo

Se encontró que uno de los cuellos de botella del sistema de alineación automática era la ejecución de ICP. Este algoritmo se llama dentro de una *View* de Django en el Editor. Esta *View* es llamada 8 veces, una vez por cada diente que se alineará. La *View* por su parte hará tantas ejecuciones de ICP como archivos de modelos en posición neutra encuentre para ese diente particular. Esto quiere decir que la ejecución de este controlador aumentará cada vez que se agreguen más modelos alineados.

Las 8 llamadas a este controlador ocurren concurrentemente, ya que el servidor esta configurado para no bloquear las request. Sin embargo esto no quiere decir que las ejecuciones se realicen en paralelo, ya que los núcleos del procesador de la máquina son de un número limitado, por lo que en realidad los núcleos disponibles se irán turnando para avanzar en todos los request.

Para solucionar este problema se hará uso del servicio de AWS Lambda para ejecutar ICP. Este servicio permite subir un trozo de código para ser ejecutado. Este código puede recibir un input y entregar un output, ambos en formato json. En este caso el código a ejecutar será ICP. La ventaja es que si se mandan a ejecutar 8 instancias de esa lambda o 80, el tiempo que se demoren en responder todas estas ejecuciones no debería aumentar, ya que se asegura que abran tantos procesadores como instancias se hayan mandado a ejecutar, debido a que AWS se encarga de gestionar los recursos y servidores para que esto sea escalable. [11]

Las Figuras 3.22 y 3.23 muestran la ejecución de las 8 request (`/align_tooth`) para dos casos, el primero con la ejecución normal en un servidor de 2 núcleos, y el segundo utilizando el mismo servidor, pero llamando AWS Lambda para ejecutar ICP. La ejecución más larga fue de 28.21 y 12.81 segundos respectivamente. Estas ejecuciones fueron hechas con el modelo de un solo paciente. Los tiempos alcanzados en la segunda ejecución no deberían variar drásticamente entonces al agregar más modelos, ya que el servicio se encargará de aumentar la cantidad de procesadores de manera dinámica según la demanda.

Name	Status	Type	Initiator	Size	Time	Waterfall
<input type="checkbox"/> constrained_jaw_regression/	200	xhr	jquery_min.js:4	314 B	430 ms	
<input type="checkbox"/> align_tooth/	200	xhr	jquery_min.js:4	553 B	24.03 s	
<input type="checkbox"/> align_tooth/	200	xhr	jquery_min.js:4	556 B	24.56 s	
<input type="checkbox"/> align_tooth/	200	xhr	jquery_min.js:4	554 B	18.23 s	
<input type="checkbox"/> align_tooth/	200	xhr	jquery_min.js:4	557 B	23.67 s	
<input type="checkbox"/> align_tooth/	200	xhr	jquery_min.js:4	554 B	18.23 s	
<input type="checkbox"/> align_tooth/	200	xhr	jquery_min.js:4	550 B	18.21 s	
<input type="checkbox"/> align_tooth/	200	xhr	jquery_min.js:4	552 B	28.21 s	
<input type="checkbox"/> align_tooth/	200	xhr	jquery_min.js:4	551 B	28.18 s	

Figura 3.22: Ejecución normal

Las Figuras 3.22 y 3.23 muestran la ejecución de las 8 request (`/align_tooth`) para dos casos, el primero con la ejecución normal en un servidor de 2 núcleos, y el segundo utilizando el mismo servidor, pero llamando AWS Lambda para ejecutar ICP. La ejecución más larga fue de 28.21 y 12.81 segundos respectivamente. Estas ejecuciones fueron hechas con el modelo de un solo paciente. Los tiempos alcanzados en la segunda ejecución no deberían variar

Name	Status	Type	Initiator	Size	Time	Waterfall	▲
<input type="checkbox"/> constrained_jaw_regression/	200	xhr	<a href="#">jquery.min.js:4</a>	297 B	1.59 s		
<input type="checkbox"/> align_tooth/	200	xhr	<a href="#">jquery.min.js:4</a>	534 B	8.86 s		
<input type="checkbox"/> align_tooth/	200	xhr	<a href="#">jquery.min.js:4</a>	537 B	8.86 s		
<input type="checkbox"/> align_tooth/	200	xhr	<a href="#">jquery.min.js:4</a>	535 B	7.94 s		
<input type="checkbox"/> align_tooth/	200	xhr	<a href="#">jquery.min.js:4</a>	537 B	8.47 s		
<input type="checkbox"/> align_tooth/	200	xhr	<a href="#">jquery.min.js:4</a>	538 B	8.27 s		
<input type="checkbox"/> align_tooth/	200	xhr	<a href="#">jquery.min.js:4</a>	534 B	8.14 s		
<input type="checkbox"/> align_tooth/	200	xhr	<a href="#">jquery.min.js:4</a>	534 B	11.69 s		
<input type="checkbox"/> align_tooth/	200	xhr	<a href="#">jquery.min.js:4</a>	535 B	12.81 s		

Figura 3.23: Ejecución utilizando AWS Lambda

drásticamente entonces al agregar más modelos, ya que el servicio se encargará de aumentar la cantidad de procesadores de manera dinámica según la demanda.

# Capítulo 4

## Integración del Algoritmo al flujo de Wizz

Para integrar el algoritmo de alineación automática se creó una aplicación web Django. Esta nueva aplicación es una fusión del *Procesador* y *Editor*. De esta forma se junta la lógica de ambos sistemas y se reduce la complejidad para el operador, ya que podrá completar la tarea de segmentar y alinear sin cambiar de página web. Además como se trata de una nueva aplicación web con base de datos propia, se mantiene independencia de los *Tratamiento Virtuales* detallados.

Se hizo deploy de esta nueva aplicación Django en el servidor del *Procesador*, debido a que este es el servidor más potente con que cuenta Wizz. Para mantener la independencia con la aplicación original (en donde se siguen segmentando los *Tratamiento Virtuales* detallados), se hizo funcionar la nueva aplicación en un puerto diferente, específicamente el puerto 9999, quedando su dominio como *procesador.wizz.cl:9999*, mientras que la aplicación original se mantiene funcionando en el puerto 80.

Los escaneos de pacientes son subidos por el escaneador a Silo. El escaneador solo tiene relación con este sistema de Wizz. Se agregó en Silo la sección para ver *Tratamiento Virtuales Preliminares* la cual provee una tabla con dos pestañas: solicitados y listos. Para esto se creó un nuevo modelo de Django como indica la figura 4.1.

En solicitados el escaneador puede ver que el *Tratamiento Virtual Preliminar* asociado al escaneo hecho por él está en curso y siendo desarrollado por un operador como indica la figura 4.2. Esta página se actualiza automáticamente cada 15 segundos, para así saber si hubo un cambio de estado y saber si el *Tratamiento Virtual Preliminar* pasó a listo. Una vez que pasa a listo en la tabla se especifica el link que dirige a la visualización del tratamiento, y el escaneador procede a mostrárselo al paciente 4.4. Además se indican como referencia el tiempo que tardó la solicitud. Esto aparece en la figura 4.3.

La view de Django que define los datos que se muestran en esta página se programó de tal manera que cada escaneador solo puede ver, tanto en solicitados como en listos, solamente los tratamientos que están asociados a los escaneos realizados por él, de forma de evitar

```

class TratamientoPreliminar(models.Model):

    SOLICITADO = "solicitado"
    LISTO = "listo"

    ESTADOS_TRAT_PRELIMINAR = (
        (SOLICITADO, SOLICITADO),
        (LISTO, LISTO)
    )

    escaneo = models.ForeignKey(to=Escaneo, on_delete=models.CASCADE, blank=False)
    estado = models.CharField(max_length=255, choices=ESTADOS_TRAT_PRELIMINAR, blank=False)
    fecha_solicitud = models.DateTimeField(default=datetime.now, blank=False)
    fecha_entrega = models.DateTimeField(blank=True, null=True)
    link = models.CharField(max_length=255, blank=True, null=True)

    def __str__(self):
        return self.escaneo.persona.nombre

```

Figura 4.1: Modelo Django que representa Tratamiento Virtual Preliminar

The screenshot displays the 'Wizz' application interface. On the left is a dark sidebar with the 'Wizz' logo and a menu including 'Mario', 'GENERAL', 'Personas', 'Agendas', 'Escaneos', 'Ventas', 'Tratamientos Preliminares' (highlighted), 'Tratamientos Virtuales', 'Tratamientos Físicos', 'Comisiones', 'Administración', and 'Cerrar Sesión'. The main content area shows a 'Solicitados' tab with 1 record and a 'Listos' tab with 2763 records. Below the tabs is a search bar and a 'Mostrar 100 registros' dropdown. A table with 5 columns (Nombre, Email, Escaneador, Fecha Solicitud, Escaneo) contains one row of data. The 'Fecha Solicitud' column shows '16 de Julio de 2019 a las 20:35' and the 'Escaneo' column shows 'solicitado'. Below the table, it says 'Mostrando registros del 1 al 1 de un total de 1 registros' and has 'Anterior' and 'Siguiente' buttons. At the bottom right, it says 'Sistema SILO. Wizz - Todos los derechos reservados.'

Figura 4.2: Pestaña solicitados

confusiones. Por otro lado los administradores ven todos los tratamientos solicitados y listos sin excepción.

Nombre	Email	Escaneador	Fecha Solicitud	Fecha Entrega	Tiempo	Link
			16 de Julio de 2019 a las 19:45	16 de Julio de 2019 a las 19:51	0:06:00	Click aquí
			16 de Julio de 2019 a las 19:38	16 de Julio de 2019 a las 19:45	0:06:31	Click aquí
			16 de Julio de 2019 a las 18:42	16 de Julio de 2019 a las 18:51	0:08:47	Click aquí
			16 de Julio de 2019 a las 17:32	16 de Julio de 2019 a las 17:38	0:06:17	Click aquí
			16 de Julio de 2019 a las 16:43	16 de Julio de 2019 a las 16:50	0:07:10	Click aquí
			16 de Julio de 2019 a las 13:18	16 de Julio de 2019 a las 13:26	0:08:06	Click aquí
			16 de Julio de 2019 a las 12:27	16 de Julio de 2019 a las 12:36	0:09:27	Click aquí
			16 de Julio de 2019 a las 11:32	16 de Julio de 2019 a las 11:45	0:12:24	Click aquí
			16 de Julio de 2019 a las 11:05	16 de Julio de 2019 a las 11:12	0:06:53	Click aquí
			16 de Julio de 2019 a las 09:38	16 de Julio de 2019 a las 09:47	0:08:06	Click aquí

Figura 4.3: Pestaña listos



Figura 4.4: Visualización del Tratamiento Virtual Preliminar

Como forma de notificar al operador que debe iniciar la fabricación de un *Tratamiento Virtual Preliminar* se implementó un mensaje de Slack automático cada vez que el escaneador sube nuevos archivos de escaneo. El código que genera el mensaje y el mensaje en sí están especificados en las figuras 4.5 y 4.6 respectivamente. La función *send\_message* envía el mensaje de slack a los operadores correspondientes que realizan el *Tratamiento Virtual Preliminar*. Una vez que el operador finaliza el proceso, el sistema notifica de forma automática a *Silo*, de forma de modificar el estado del *Tratamiento Virtual Preliminar* de solicitado a listo.

```

tratamiento_preliminar = TratamientoPreliminar(
    estado=TratamientoPreliminar.SOLICITADO,
    escaneo=escaneo
)
tratamiento_preliminar.save()
try:
    mensaje = '**** NUEVO ESCANEO SUBIDO ****\nPACIENTE: ' + persona.nombre\
        + '\nEMAIL: ' + persona.email\
        + '\nOBSERVACIONES: ' + escaneo.observaciones\
        + '\nESCANeadOR: ' + escaneo.escaneador.user.get_full_name()
    send_message('G', mensaje)
    send_message('I', mensaje)
    send_message('T', mensaje)
    r = requests.post(settings.FAST_TREATMENT_URL, data={'escaneo_id': escaneo.id})
except:
    pass

```

Figura 4.5: Código para generar mensaje de Slack

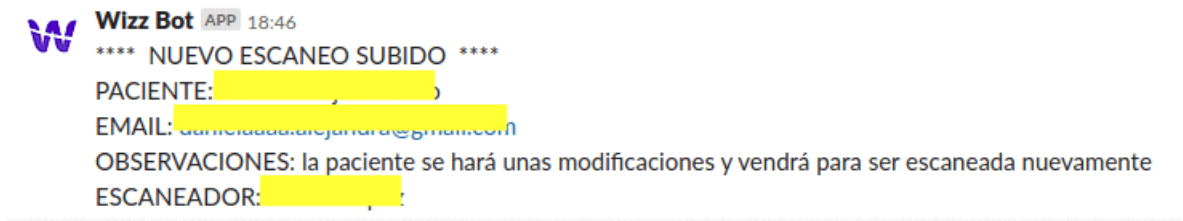


Figura 4.6: Mensaje de Slack que recibe el operador

# Capítulo 5

## Evaluación Experimental

### 5.1. Pruebas del algoritmo de alineación

Los resultados mostrados a continuación corresponden a ejecuciones del algoritmo de alineación automática, sin editar los modelos posteriormente. Las pruebas se realizaron habiendo en el servidor guardados los modelos de dientes en posición neutra de dos pacientes para la aplicación de ICP.

Luego de aplicado el algoritmo, se ejecutó el algoritmo de deformación de encía, el cual ya existía anteriormente antes del desarrollo de este trabajo. El algoritmo utiliza una funcionalidad de la librería 3D libigl de “deformación biarmónica”. Dado un cuerpo y un subconjunto de vértices del cuerpo, se especifican nuevas posiciones para este subconjunto de vértices, para finalmente obtener el cuerpo deformado [12]. Los vértices móviles en este caso son los puntos de contacto entre el diente y la encía. Se puede acceder a esta funcionalidad por medio de un botón en el *Editor*.

El tiempo de ejecución promedio del algoritmo de alineación en estos casos fue de 23.6 segundos.

Para ejecutar el algoritmo basta con presionar un botón: alinear maxilar (arcada superior) o alinear mandíbula (arcada inferior), y luego hacer click en el primer premolar del lado izquierdo de la arcada correspondiente.

Se agregó el plugin jQuery BlockUI <http://malsup.com/jquery/block/>, el cual bloquea la interfaz del *Editor* mientras se ejecuta el algoritmo, mostrando a su vez un spinner.

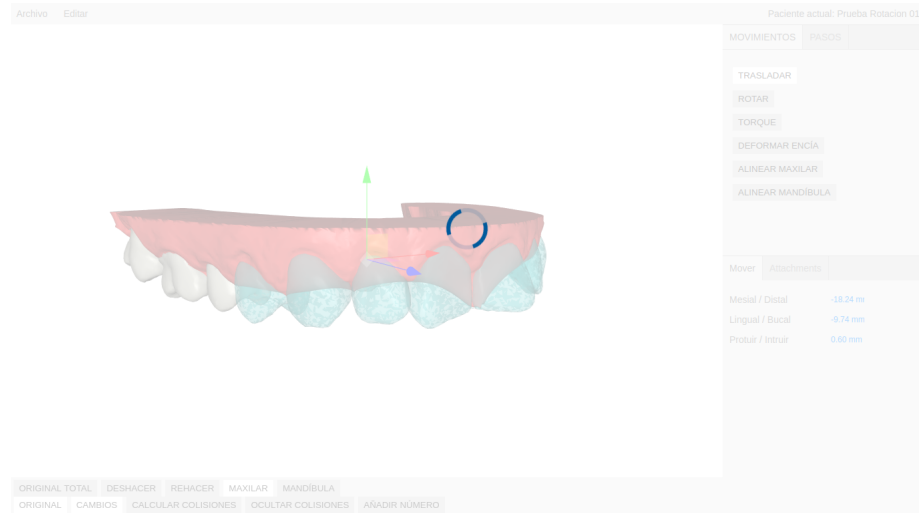


Figura 5.1: Interfaz del *Editor* bloqueada durante ejecución del algoritmo

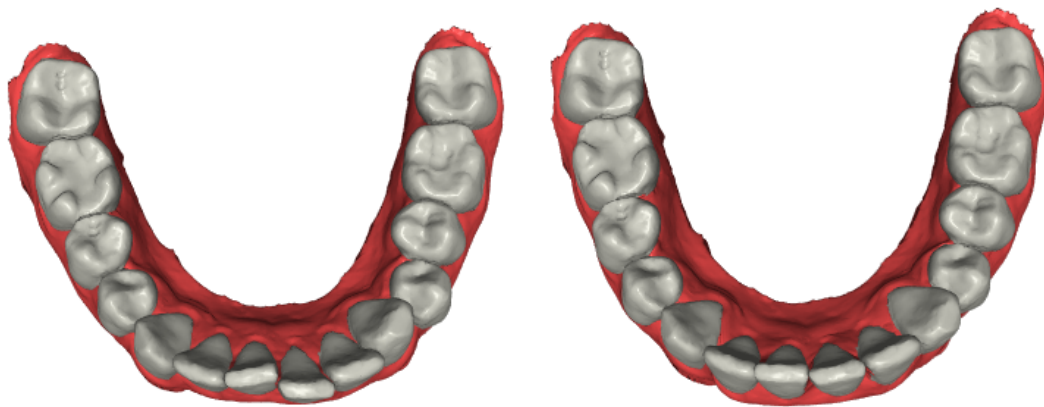


Figura 5.2:

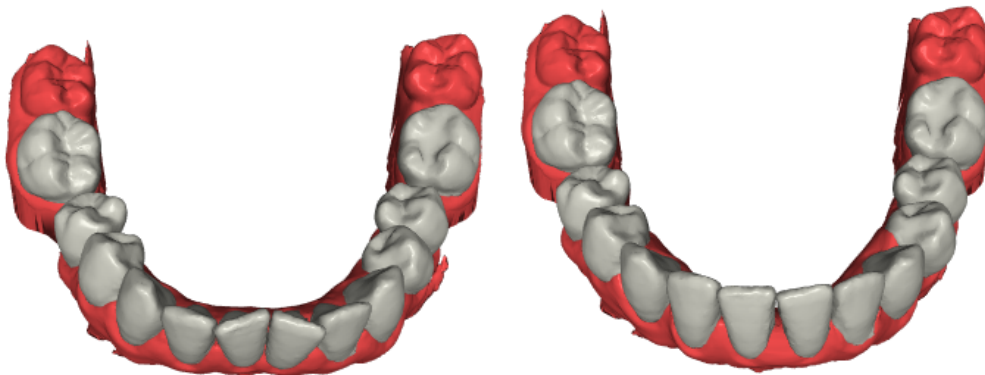


Figura 5.3:





Figura 5.4:

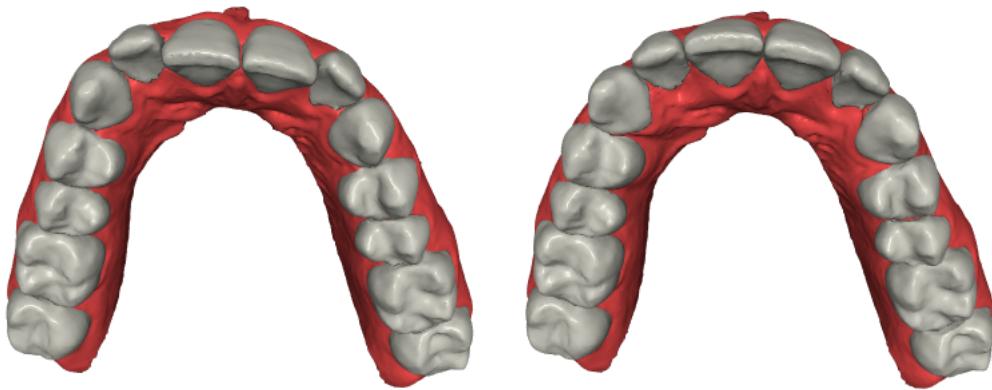


Figura 5.5:

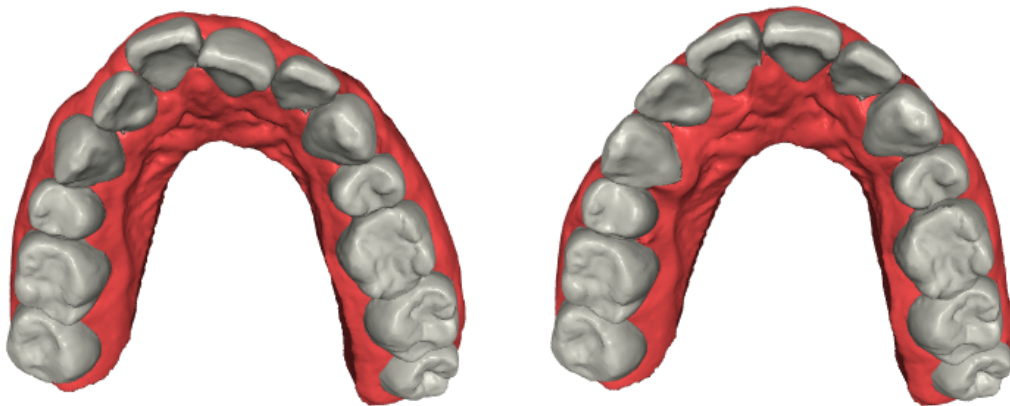


Figura 5.6:

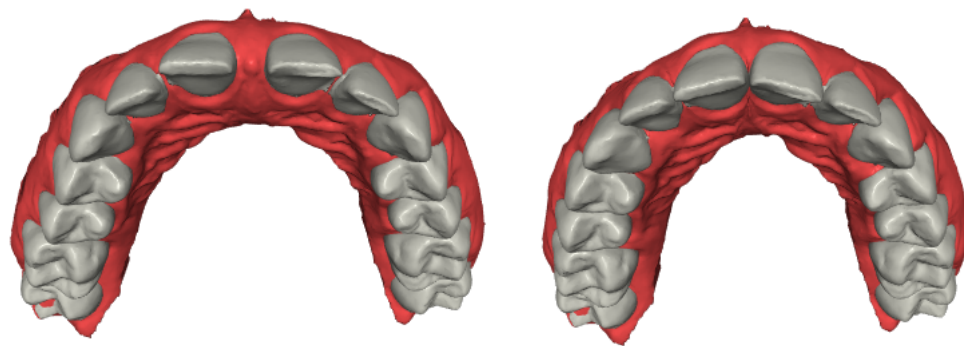


Figura 5.7:

## 5.2. Correctitud de la alineación alcanzada

Según lo expresado en los objetivos de este trabajo, el sistema busca obtener una sonrisa más estética de la que el paciente tiene actualmente. Esta es una propiedad difícil de expresar incluso para los operadores que realizan este trabajo actualmente. Pero a pesar de que para ellos es una propiedad difícil de expresar, se asume que ellos realizan el trabajo correctamente, y llegan siempre al resultado más estético posible.

Al ser este un sistema que intenta replicar el trabajo de los operadores, una forma de calificar la calidad de la alineación automática es medir el movimiento que tuvo que hacer posteriormente el operador para corregir el resultado obtenido y llegar al resultado que a él le parecía correcto. Este movimiento se puede expresar como la diferencia entre el estado final e inicial, para las traslaciones y rotaciones aplicadas manualmente a cada diente.

Los pacientes que llegan a Wizz presentan casos de variada gravedad, algunos requieren movimientos muy leves, y otros movimientos más pronunciados. Medir el movimiento que tuvo que hacerse para corregir la alineación automática, sin ponerla en el contexto de la gravedad del paciente particular, puede ser engañoso. La prueba ideal sería comparar el movimiento al hacer el trabajo completamente manual versus el movimiento al hacer el trabajo asistido por la alineación automática, considerando el mismo paciente en ambos casos.

Una posibilidad para medir esto es considerar los pacientes que cuentan tanto con *Tratamiento Virtual Preliminar* (el cual es asistido por la alineación automática), como con *Tratamiento Virtual* (que se hace completamente manual) terminados, es decir los pacientes que han comprado el tratamiento.

Se definen las siguientes variables, considerando los 8 dientes que mueve el algoritmo de alineación automática:

- *media\_traslacion*: media del cambio en la traslación, calculada como la distancia euclidiana entre la posición final e inicial del diente

$$\left(\sum_{i=0}^7 \sqrt{(position\_fin[i].x - position\_ini[i].x)^2 + (position\_fin[i].y - position\_ini[i].y)^2 + (position\_fin[i].z - position\_ini[i].z)^2}\right)/8$$

- *media\_rot\_x*: media del cambio en la rotación con respecto al eje  $x$

$$\left(\sum_{i=0}^7 |rotation\_fin[i].x - rotation\_ini[i].x|\right)/8$$

- *media\_rot\_y*: media del cambio en la rotación con respecto al eje  $y$

$$\left(\sum_{i=0}^7 |rotation\_fin[i].y - rotation\_ini[i].y|\right)/8$$

- *media\_rot\_z*: media del cambio en la rotación con respecto al eje  $z$

$$\left(\sum_{i=0}^7 |rotation\_fin[i].z - rotation\_ini[i].z|\right)/8$$

Definidas estas variables se procedió a medir el movimiento manual hecho en el Tratamiento Virtual y el Tratamiento Virtual Preliminar, para 10 pacientes reales. En ambos casos el tratamiento fue realizado por operadores de Wizz. La hipótesis es que el movimiento manual en el *Tratamiento Virtual Preliminar* debería ser mucho menor ya que se tratan solo de correcciones realizadas a lo entregado por la alineación automática. La figura 5.8 muestra el resultado de estas mediciones. La traslación esta medida en milímetros y la rotación en grados.

#	tratamiento_id	Movimiento Manual Tratamiento Virtual				Movimiento Manual Tratamiento Virtual Preliminar			
		media_traslacion	media_rot_x	media_rot_y	media_rot_z	media_traslacion*	media_rot_x*	media_rot_y*	media_rot_z*
1	3515	1,139	10,018	6,266	11,076	0,297	0,423	0,03	1,776
2	3510	0,411	3,743	6,271	2,973	0,061	3,205	0,055	3,070
3	3505	1,029	6,226	3,355	5,290	0,094	0,124	1,542	0,270
4	3508	1,088	11,369	7,141	11,511	0,010	3,014	0,557	3,278
5	3507	1,112	3,893	8,064	6,678	0,174	0,796	0,183	1,763
6	3506	0,777	16,162	5,997	14,292	0,126	0,292	0,008	2,070
7	3498	0,662	10,118	8,642	6,351	0,000	1,517	0,151	1,377
8	3503	0,779	11,885	3,595	12,416	0,000	3,389	0,13	3,328
9	3500	0,821	2,925	2,608	3,829	0,060	2,977	0,453	2,865
10	3497	0,689	4,195	3,897	6,617	0,163	0,841	2,071	2,431

Figura 5.8: Comparación entre el movimiento manual realizado en Tratamiento Virtual y Tratamiento Virtual Preliminar

#	tratamiento_id	media_traslacion*/media_traslacion	media_rot_x*/media_rot_x	media_rot_y*/media_rot_y	media_rot_z*/media_rot_z
1	3515		26%	4%	0%
2	3510		15%	86%	1%
3	3505		9%	2%	46%
4	3508		1%	27%	8%
5	3507		16%	20%	2%
6	3506		16%	2%	0%
7	3498		0%	15%	2%
8	3503		0%	29%	4%
9	3500		7%	102%	17%
10	3497		24%	20%	53%

Figura 5.9: Porcentaje del movimiento que se realiza en el caso del *Tratamiento Virtual Preliminar* con respecto al *Tratamiento Virtual*

Podemos ver como solo para dos pacientes y solo para uno de los movimientos medidos, el operador tuvo que hacer más movimiento que en el caso completamente manual, concluyendo que por lo general el operador hace menos movimiento. Esto implica que el resultado de la alineación automática logra un estado de los dientes más estético que el estado original a criterio del operador, cuyo trabajo se pretende facilitar.

```

function getMovimiento(jaw, initTeeth){
    var traslacion = 0;
    var rotacionX = 0;
    var rotacionY = 0;
    var rotacionZ = 0;
    for(var i = numTeeth; i < initTeeth + 8; i++){
        var tooth = getTeethByNumber(jaw, i);
        var tooth_original = getOriginalTeethByNumber(jaw, i);
        traslacion += Math.sqrt(
            x: Math.pow(x: tooth.position.x - tooth_original.position.x, y: 2)
            + Math.pow(x: tooth.position.y - tooth_original.position.y, y: 2)
            + Math.pow(x: tooth.position.z - tooth_original.position.z, y: 2));
        rotacionX += Math.abs(x: THREE.Math.radToDeg(tooth.rotation.x)
            - THREE.Math.radToDeg(tooth_original.rotation.x));
        rotacionY += Math.abs(x: THREE.Math.radToDeg(tooth.rotation.y)
            - THREE.Math.radToDeg(tooth_original.rotation.y));
        rotacionZ += Math.abs(x: THREE.Math.radToDeg(tooth.rotation.z)
            - THREE.Math.radToDeg(tooth_original.rotation.z));
    }
    return { 'traslacion_media': traslacion/8,
            'rotacion_x_media': rotacionX/8,
            'rotacion_y_media': rotacionY/8,
            'rotacion_z_media': rotacionZ/8};
}

```

Figura 5.10: Parte del código para obtener el movimiento realizado

### 5.3. Pruebas del proceso completo de fabricación de Tratamiento Virtual Preliminar

Dentro del proceso de fabricación del Tratamiento Virtual Preliminar la segmentación se siguió haciendo de forma manual, pero con modelos con una resolución menor que en los Tratamiento Virtuales detallados, por lo que la performance mejoró bastante, además el operador deja pasar pequeñas imperfecciones sin repetir el procedimiento. Se analizaron 10 segmentaciones. El proceso de segmentación dura 5 minutos en el caso usual y hasta 8 minutos en los casos más complejos, resultando en promedio en 6.34 minutos de duración.

Luego se tomó el tiempo a la alineación, siendo esta la suma de la parte automática y los movimientos de corrección que realiza posteriormente el operador. Se analizaron 10 alineaciones, las cuales variaron entre 2 a 5 minutos, con un promedio de 3.79 minutos.

En total ambos procesos demoraron 10.13 minutos en promedio. Sin embargo estos eran casos donde no hubo peticiones de tratamiento concurrentes. Como se vio en el capítulo anterior, en Silo se puede ver el tiempo que demoró en entregarse Tratamiento Virtual Preliminar desde su solicitud, siendo la mayoría coincidentes con el valor de 10 minutos, pero existen otros que demoraron 20 minutos y 30. Esto sucede cuando hay solicitudes de tratamientos concurrentes, es decir se genera una solicitud mientras aún se está trabajando en una solicitud anterior.

# Capítulo 6

## Conclusiones y Trabajo Futuro

Se concluye que el algoritmo desarrollado logra acercarse a una posición de los dientes más estética debido a que el operador hace menos movimientos en la dentadura en contraste con hacer todo el proceso de alineación de forma manual. Esto no quiere decir necesariamente que ambas soluciones se parezcan, por lo que aún es un riesgo aplicar este algoritmo como una ayuda para realizar el *Tratamiento Virtual* detallado en el que se basa el *Tratamiento Físico*, principalmente porque este algoritmo no intenta minimizar el movimiento de los dientes con respecto al estado inicial.

Un operador aparte de proponer una posición más estética de los dientes esta a su vez minimizando el movimiento que hacen los dientes con respecto a la posición inicial, porque esto reduce la cantidad de alineadores que tendrá el tratamiento y en consecuencia la duración de este. Una vez que se aplica el algoritmo automatizado desarrollado para este proceso el operador pierde referencia de la posición inicial real de los dientes, y solo intenta hacer pequeños ajustes para llegar con la mínima cantidad de movimiento al estado final que a él le parece más estético.

Si bien los tiempos de duración del proceso completo de fabricación de un *Tratamiento Virtual Preliminar* son satisfactorios, ya que están dentro de lo que un paciente está dispuesto a esperar, la escalabilidad del sistema se ve afectada por la necesaria dependencia de un operador. Un operador podrá responder al tiempo esperado y medido siempre cuando las solicitudes de tratamiento lleguen a una frecuencia constante similar al tiempo que la fabricación demora, o sea una solicitud cada 15 minutos aproximadamente.

Esta frecuencia no siempre se cumple debido a que un centro de escaneo puede ejecutar un escaneo cada media hora, pero existen ya más de cinco centros de escaneo. Wizz por su parte pretende seguir abriendo más. Por lo que se producirán horas de alta afluencia, en que pueden llegar 3 solicitudes de tratamiento en un minuto, u horarios de baja afluencia en que no llega ninguna solicitud.

Sin embargo el cambio del flujo de negocio para agregar la entidad de *Tratamiento Virtual Preliminar*, siempre será un mejor escenario que el que se tenía anteriormente, a pesar de que no necesariamente los pacientes verán este tratamiento el 100 % de las veces en el mismo

centro de escaneo, y tendrán que recibirlo posteriormente vía correo electrónico. Esto porque de todas formas es menos costoso que la fabricación de un *Tratamiento Virtual* detallado, y en parte eso tiene que ver con la efectividad del algoritmo propuesto en esta memoria.

Como trabajo futuro se podría intentar prescindir de los operadores en la fabricación de *Tratamiento Virtual Preliminar* para poder hacer crecer el negocio por medio de la apertura de más centros de escaneo, sin necesidad de hacer crecer el personal. Para esto es necesario primero automatizar el proceso de segmentación. Luego mejorar el algoritmo aquí descrito de tal manera que no sea necesario hacer correcciones manuales a la alineación.

Un buen approach en vez de prescindir completamente de un humano (lo que sería bastante difícil por la complejidad del problema) sería desarrollar algoritmos tanto en la segmentación y alineación que generen varias *propuestas* de solución. De esta manera el operador tendría solo que dedicarse a elegir la propuesta de segmentación y alineación generada por el sistema que le parezca más correcta, lo que reduciría al mínimo la interacción del operador con el sistema y a su vez dándole la posibilidad de trabajar en más de un tratamiento al mismo tiempo. El hecho de que los algoritmos propongan un set de resultados y no solo uno, podría reducir de forma importante la complejidad del código.

Si bien el algoritmo aquí descrito fue funcional para solucionar el problema, el perfeccionamiento de este podría ser algo costoso, y quizás si se quieren llegar a mejores resultados sería conveniente pensar en una estrategia de solución completamente diferente, por ejemplo haciendo uso de machine learning. El año 2018 salió la primera publicación en que se usaba un algoritmo de estas características para realizar el proceso de segmentación dental de forma 100% automática [13], por lo que la aplicación de estos algoritmos en modelos dentales 3d es algo efectivo, pero a su vez incipiente, aumentando la incertidumbre en el éxito del desarrollo de un software de esas características.

# Bibliografía

- [1] Hasse, Susan. (1975). *Polynomial and Catenary Curve Fits to Human Dental Arches*. Journal of dental research 54(6). 1124-32.
- [2] Noroozi, Hassan Hosseinzadeh Nik, Tahereh Saeeda, R. (2001). *The Dental Arch Form Revisited*. The Angle orthodontist. 71(5). 386-9.
- [3] Besl, Paul McKay, H.D.. (1992). *A method for registration of 3-D shapes*. IEEE Transactions on Pattern Analysis and Machine Intelligence. 14(2). 239-256.
- [4] Chen, Yang Medioni, Gérard. (1992). *Object Modeling by Registration of Multiple Range Images*. IEEE International Conference on Robotics and Automation. 10(3). 145-155.
- [5] Strutz, Tilo. (2016). *Data Fitting and Uncertainty (A practical introduction to weighted least squares and beyond)* (2nd edition).
- [6] Faq de Django. Junio 2019. <https://docs.djangoproject.com/es/2.1/faq/>
- [7] ¿Por que Slack? Como funciona. Junio 2019 <https://slack.com/intl/es-cl/features>
- [8] Slack Web API. Junio 2019 <https://api.slack.com/web>
- [9] three.js documentation. Junio 2019 <https://threejs.org/docs/>
- [10] three.js github. Junio 2019 <https://github.com/mrdoob/three.js>
- [11] Características de AWS Lambda. Junio 2019. <https://aws.amazon.com/es/lambda/features/>
- [12] libigl tutorial: Biharmonic Deformation. Agosto 2019. <https://libigl.github.io/tutorial/#biharmonic-deformation>
- [13] X. Xu, C. Liu Y. Zheng. (2018). *3D Tooth Segmentation and Labeling Using Deep Convolutional Neural Networks*. IEEE Transactions on Visualization and Computer Graphics. 25(7). 2336-2348.