E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Técnicas de análisis y evaluación de procesos de reconstrucción 3D en el ámbito de la localización por visión

Grado en Ingeniería
en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Ugarte Albizu, Ramón Jesús

Zalba Olcóz, Jeser

Pamplona, 30 de junio de 2015

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Agradecimientos

Agradecer en primer lugar a mis padres por darme el apoyo y los medios para poder estudiar lo que me gusta. Y a mis hermanos y todos los familiares que han estado ahí.

Al Dr. David Oyarzun Laura, director del área de Computación Gráfica e Interacción de la Fundación VIcomtech-IK4, por permitirme realizar este TFG en Vicomtech y toda su ayuda y confianza.

A Jeser Zalba Olcoz, mi director de TFG, por su confianza depositada en mí y toda su ayuda y consejo durante el desarrollo de este TFG.

Agradecer de manera especial a Nagore Barrena Orueechebarria por su inestimable tutela, todas las horas dedicadas a enseñarme todo lo he necesitado saber y toda la ayuda que me ha ofrecido durante mi estancia en Vicomtech.

Por último, a mis compañeros de departamento y otros en Vicomtech que me han permitido formar parte de esta gran familia desde un primer momento, en especial a Victor, Rubén, Javi, Unai, Hugo, Sara, Andoni, Andrés, Jairo y un largo etc.

Resumen

El presente trabajo se centra en el análisis y evaluación de procesos localización por visión artificial en al ámbito de la Realidad Aumentada (AR).

El objetivo último de las aplicaciones de AR es determinar de forma precisa la posición del usuario para poder proyectar, desde su perspectiva, elementos virtuales sobre el mundo real de manera alineada. Con este fin, se hace necesario basar la localización en el reconocimiento de elementos del entorno.

El presente trabajo analiza en profundidad la mejora de precisión en la localización con el uso de la información aportada por la curvatura predominante de cada uno de los puntos que componen los elementos del entorno.

ÍNDICE

Lista de figuras

Lista de Tablas

Terminología

- **Frame**: Imagen extraída de un video.

- **Tracking**: En castellano, “seguimiento”. Técnica capaz de seguir el movimiento de la cámara entre las diferentes *frames* de un video.

- **Entorno**: Hace referencia al lugar físico sobre el que se realizan los procesos de reconstrucción y *tracking*.

- **Renderizar**: Método por el cual se crean imágenes 2D de un entorno digital 3D creado por ordenador.

- **Online**: Procesos realizados en tiempo real.

- **Offline**: Procesos realizados fuera del tiempo real.

- **Parche**: Conjunto de pixeles tomados de una imagen más grande generalmente en torno a un punto característico.

- **Template Matching**: Técnica capaz de calcular la similitud entre 2 imágenes comparándolas mediante diferentes métodos.

- **Feature**: En castellano, “característica”. En este caso se refiere a un punto 2D dentro de una imagen, el cual tiene una característica que permiten diferenciarle del resto de puntos.

- **Vector Normal**: Vector de longitud unidad que indica de manera perpendicular la orientación de una superficie.

- **KeyFrame**: Imagen de referencia de un entorno utilizada para su reconstrucción 3D

- **Optical Flow**: Técnica de seguimiento del flujo de movimiento de los diferentes puntos entre los diferentes *frames* de un video.

Parte Primera

INTRODUCCIÓN

1.1 Motivación y Objetivos

Debido al desarrollo llevado a cabo en la última década en el ámbito de las aplicaciones y servicios para teléfonos móviles, tabletas y otros gadgets portátiles, se han hecho muy populares aplicaciones que hacen uso de la cámara integrada en estos dispositivos. Muchas de estas aplicaciones se basan en la realidad aumentada.

La realidad aumentada (en adelante AR) se basa en el concepto de añadir información digital al entorno en que nos encontremos, alineando la imagen real con la generada por parte de la aplicación.

La AR se considera como la evolución de la realidad virtual (en adelante VR). La tecnología VR, sumerge al usuario en un mundo completamente artificial. Mientras el usuario está inmerso en dicho mundo virtual, no tiene ningún tipo de noción sobre el entorno real que lo rodea. Sin embargo, la tecnología AR, permite al usuario ver el mundo real e imágenes artificiales/virtuales son superpuestas en dicho mundo; lo real y lo virtual coexisten en un mismo espacio.

El trabajo descrito en [1] describe el concepto expresado mediante el uso del denominado *Reality-Virtuality Continuum* (*Ilustración 1*), una escala que oscila entre lo completamente real y lo completamente virtual. El área entre los dos extremos dónde ambos, lo real y lo virtual son mixtos, se denomina *mixed reality*. La AR es el área donde lo virtual se sobrepone a lo real, y el área de *augmented virtuality* es el área donde lo real se sobrepone a lo virtual.

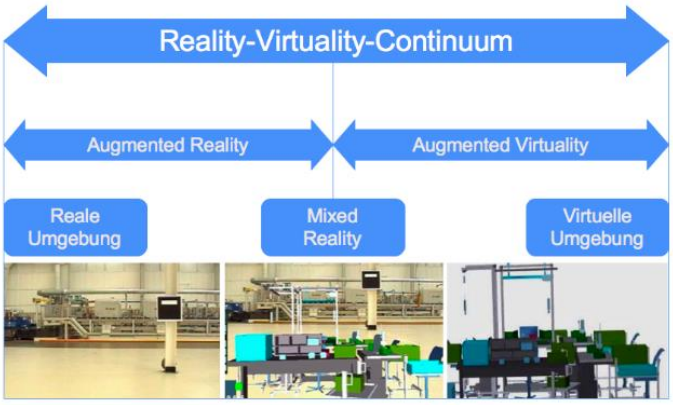


Ilustración 1 Reality- Virtuality Continuum.

De acuerdo al trabajo presentado en [2], las aplicaciones de AR se definen con las siguientes características: combina elementos reales y virtuales, que interactúan en tiempo-real y deben estar registrados en 3D. Por tanto, tal y como se ha mencionado con anterioridad lo real y lo virtual debe coexistir en un mismo espacio; los elementos

virtuales son modelos 2D/3D que deben ser añadidos en una posición concreta del entorno en tiempo real.

El usuario de la aplicación de AR, debe ver una imagen en donde el entorno real está capturado, y donde se visualizan proyectados uno o más elementos virtuales. Se observa un ejemplo de lo mencionado en la *Ilustración 2*; el esqueleto es un elemento virtual en un entorno real. Es por ello, que el objetivo principal de una aplicación de AR, es posicionar el elemento virtual en una posición concreta del entorno real de modo continuado. Con dicho objetivo, la localización del objeto/entorno debe identificarse de manera continua, incluso cuando el objeto o la cámara están en movimiento. A este proceso se le denomina *tracking.*

Ilustración 2: Ejemplo de aplicación de AR.

Teniendo en cuenta las características mencionadas, un sistema de AR requiere de los siguientes componentes: una cámara que capture el mundo real, un procesador para realizar los cálculos necesarios y un *display* (pantalla, proyector, etc.) donde la imagen del entorno real con las proyecciones de los elementos virtuales sea *renderizada*.

Como se ha mencionado al inicio, la AR se está popularizando gracias a la emergencia en el mercado de las nuevas generaciones de dispositivos móviles. Dichos dispositivos tienen una buena conexión a Internet, así como procesadores con una buena capacidad computacional, que le otorgan capacidades de multimedia avanzadas. Por otro lado, también están equipadas con cámaras de una buena calidad. Además, con el objetivo de sobreponer los elementos virtuales, es común usar la información de posición y orientación dada por el GPS y los sensores del dispositivo, como acelerómetros o giróscopos.

Layar, por ejemplo, es uno de los buscadores más exitosos para AR. El funcionamiento básico de Layar consiste en proporcionar la información referente a diferentes lugares. El buscador es muy usado en el campo del turismo, proporcionando mucha información útil para el usuario sobre los lugares en los que se encuentre. Por ejemplo, información histórica sobre determinados edificios, información acerca del transporte público, etc. En la *Ilustración 3* se puede observar un ejemplo de la aplicación. Layar funciona combinando el uso de la cámara, la brújula y el sistema GPS para realizar un *tracking* del usuario. Después localiza en su base de datos esas coordenadas geográficas y muestra en la pantalla la información superpuesta sobre la imagen de la cámara.

Ilustración 3: Ejemplo aplicación Layar.

Cuando se utilizan los sensores (acelerómetro y brújula) y GPS para localizar al dispositivo, la información de AR normalmente está limitada a etiquetas 2D. La precisión en la lectura de esos sensores no es suficiente para introducir elementos 3D alineados correctamente con la imagen.

El uso de marcadores para realizar el *tracking* es un método muy común en las aplicaciones móviles de AR. Los marcadores son generalmente impresiones en papel colocadas en el entorno. Los patrones se conocen previamente y cuando son captados con la cámara, se reconocen gracias a ese entrenamiento previo. La estimación de la posición de la cámara utilizando esta técnica es una tarea sencilla para los dispositivos. Muchos algoritmos de visión por ordenador hacen uso de marcadores. En la *Ilustración 4* se muestra un tipo determinado de marcador.

Ilustración 4: Marcador ARToolkit.

Existe un gran número de aplicaciones de AR que utilizan marcadores para el *tracking*. En la industria del videojuego también se han utilizado los marcadores en muchos productos con la intención de crear unos resultados más atractivos. “Invizimals”, el juego para PSP (Play Station Portable, Sony) es uno de los juegos más famosos que hacen uso de estas técnicas. Como se ve en la *Ilustración 5*, el *tracking* se realiza utilizando un marcador cuadrado. En el juego, el escenario es el entorno elegido por el usuario. El resto de componentes en el juego, personajes, objetos, etc. Son elementos virtuales añadidos al escenario real.

Ilustración 5: Ejemplo aplicación Invizimals.

El *tracking* basado en marcadores es muy rápido y preciso. Sin embargo, los marcadores han de estar pegados en diferentes lugares de entorno. Esta característica lo hace poco práctico. Por esa razón, se ha hecho popular otro tipo de *tracking* basado en marcadores.

El marcador de la *Ilustración 4* pertenece a un tipo llamados marcadores ARToolkit. ARToolkit es una librería creada para desarrolladores de aplicaciones de AR. Se utilizan para funciones de *tracking*.

Para eliminar la proliferación de marcadores artificiales dentro de la escena, se utilizan otro tipo de marcadores llamados marcadores naturales. La idea detrás de este concepto es la de usar marcadores pertenecientes al entorno para estimar la posición de la cámara en el entorno de manera fácil y precisa. Dado que los elementos utilizados para el *tracking* existen ya dentro de la escena, han de ser seleccionados y pre-procesados antes de utilizarlos como marcadores naturales.

La marca de juguetes LEGO utiliza aplicaciones de AR con finalidades comerciales. La aplicación hace uso de aplicaciones de *tracking* con marcadores naturales. La imagen que proporciona cualquier caja de Juegos LEGO se puede utilizar como un marcador natural. Así pues, cualquier tipo de elemento virtual puede ser añadido al entorno real. Cuando la caja del juego es reconocida por la cámara del dispositivo móvil, los elementos del juego LEGO cobran vida como modelos virtuales. La *Ilustración 6* muestra un ejemplo de la aplicación LEGO.

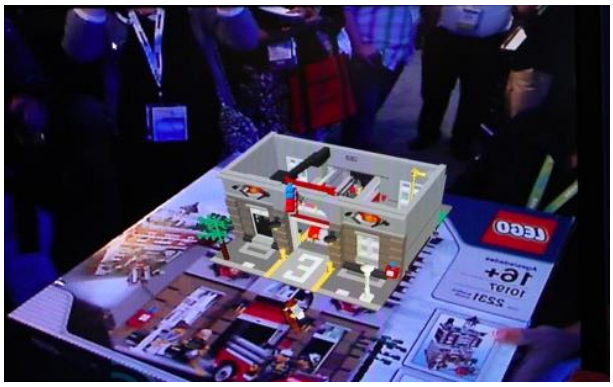


Ilustración 6: Ejemplo aplicación LEGO.

Sin embargo, hay muchos métodos precisos de *tracking* que no necesitan un entrenamiento previo y que hacen uso de técnicas de visión por computador con el fin de localizar el objeto. Se identifican dos retos en estos métodos: por un lado, necesitan la información 3D del entorno a modo de mapa con el fin de localizarse en ellos. Esa información 3D debe ser calculada por el sistema. Por otro lado, los sistemas de *tracking*

deben localizar el dispositivo en dicho mapa haciendo uso de la información que proveen las imágenes del entorno capturadas por las cámaras de los dispositivos. Es decir, es imprescindible establecer una relación entre los puntos 3D que componen el mapa 3D y sus correspondientes proyecciones (2D) en las imágenes.

Respecto al primero de los retos mencionados, Este tipo de aplicaciones necesita de una gran capacidad de cómputo y los dispositivos móviles están limitados en ese aspecto, ya que han de llegar a un equilibro entre rendimiento y autonomía.

Es por ello que se trata de minimizar ese problema realizando las operaciones de cálculo previas a detección de manera *offline*, dejando al dispositivo móvil solamente la tarea del *tracking* pura.

En cuanto al segundo de los retos, establecer relación entre los puntos 3D del entorno y sus correspondientes 2D en las imágenes, existen diversas soluciones en el estado del arte, pero todas ellas presentan algún problema. La solución más extendida se centra en el uso de métodos denominados *Template Matching,* por ser los más precisos. En esta técnica, se comparan partes de las imágenes de entrada con las imágenes usadas para hallar el mapa 3D. Del resultado de dicha comparación se establece la relación entre proyecciones y puntos 3D.

A pesar de popularidad de esta técnica y de su precisión existen ciertas limitaciones, que se tratarán de minimizar gracias a las tareas y resultados obtenidos en este Trabajo fin de Grado (de aquí en adelante TFG).

1.2 Contribuciones

Durante la Sección 1.1 se han expuesto los diferentes métodos de *tracking* y sus limitaciones. El principal objetivo de este TFG es demostrar la mejora de los procesos de *tracking* mediante técnicas de *Template Matching* pero en este caso, teniendo en cuenta la curvatura predominante de los puntos reconstruidos.

Con dicho propósito, los principales hitos llevados a cabo durante el desarrollo de este TFG se enumeran a continuación:

- Creación de un entorno 3D controlado en el que se conoce con exactitud la geometría y localización de todos sus elementos.

- Extracción de imágenes de referencia del entorno de las cuales se han extraído puntos característicos (proyecciones 2D).

- Proyección inversa de esos puntos característicos de vuelta al mundo 3D para extracción de nube de puntos junto a sus vectores normales (*Raycaster*).

- Adición progresiva de ruido a los vectores normales.

- Aplicación del algoritmo mediante *template matching* para reconstrucciones con ruido.

- Análisis estadístico de los resultados obtenidos.

1.2.1 Planificación

La duración total del trabajo ha comprendido 19 semanas con la siguiente distribución:

Tarea 1 “Lectura recomendada”	Semana 1-2
Tarea 2 “Adaptación al método de trabajo”	Semana 3-4
Tarea 3 “Creación de escena 3D”	Semana 4-5
Tarea 4 “Extracción de *featured points*”	Semana 6
Tarea 5 “Cálculo de posición y vector normal de *featured points* en la escena (*Raycaster*)”	Semana 7-10
Tarea 6 “*Renderización* de imágenes sintéticas”	Semana 11
Tarea 7 “Adición de ruido a los vectores normales”	Semana 12-13
Tarea 8 “Ejecución del algoritmo (WaPT)”	Semana 14-15
Tarea 9 “Análisis”	Semana 16-17
Tarea 10 “Redacción”	Semana 16-19

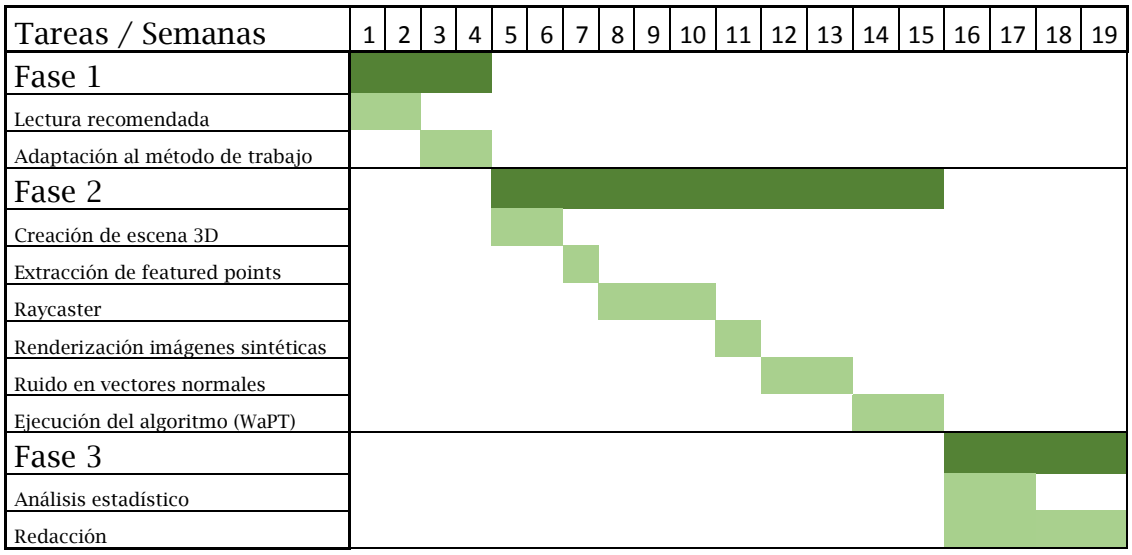


Tabla 1 Diagrama de Gantt

1.3 Esbozo del Trabajo Fin de Grado

El contenido de este documento está dividido en siete capítulos. En el primero de ellos, se realiza una introducción de la tecnología de AR así como de los objetivos y motivación del presente trabajo. Por otro lado, la planificación realizada para el desarrollo del trabajo también está expresada en dicho capítulo. En el capítulo 2, se presentan las diferentes técnicas de visión por computador usadas para los procesos de tracking y su uso en sistemas de AR. En lo que al capítulo 3 se refiere, éste describe en profundidad el algoritmo "WaPT", que será evaluado y usado para demostrar la hipótesis principal de este trabajo fin de grado. En el capítulo 5 las tareas realizadas para realizar la demostración son enumeradas y detalladas en profundidad. Los resultados obtenidos, de las tareas y experimentos realizados son expuestos en el capítulo 6 y las conclusiones obtenidas sin embargo se encuentran expresadas en el capítulo 7.

Antecedentes Capítulo 2

Las aplicaciones de AR para dispositivos móviles normalmente hacen uso de la posición y orientación proporcionada por su GPS, acelerómetro o brújula para superponer la información. Sin embargo, la precisión de lectura de esos sensores no es suficiente para insertar el contenido 3D correctamente dentro de la imagen.

Por otro lado, los métodos de *tracking* basados en marcadores aseguran una alta precisión en el proceso de *tracking*, pero se necesita información previa y en ciertos casos se contamina la escena colocando marcadores dentro de ella.

No obstante, existen determinados algoritmos de visión que son capaces de estimar la posición y la orientación del dispositivo de forma correcta sin contaminar el entorno.

Es importante conocer cómo funciona el modelo de cámara pin-hole y entender cómo se realiza el proceso de *tracking*, con el fin de indagar en dichas técnicas de visión artificial.

Este capítulo proporciona una introducción al mencionado modelo de cámara y a las técnicas de *tracking* por visión artificial para su uso en aplicaciones de AR.

2.1 Modelo de Cámara

El modelo de cámara describe la relación entre los puntos 3D en un escenario real y sus proyecciones en el plano de imagen. Existen muchos tipos de cámara diferentes, pero el más utilizado y representativo es el modelo "pin-hole", *Ilustración 7*.

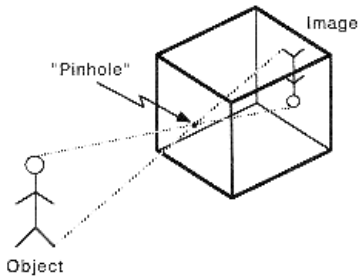


Ilustración 7 Cámara Pin-Hole

Los elementos que definen este tipo de cámara son:

- Eje óptico.

- Plano de imagen.

- Centro óptico.

- Distancia focal.

El eje óptico es la línea que parte de la cámara (centro óptico) y perpendicularmente atraviesa al plano de imagen en su punto central (*Ilustración 8*). Extrapolando, la proyección 2D de un punto del espacio $(X, Y, Z)^T$ en el plano de imagen $(x, y)^T$ se define como la intersección entre el plano de imagen y la línea que une el punto 3D con el centro óptico.

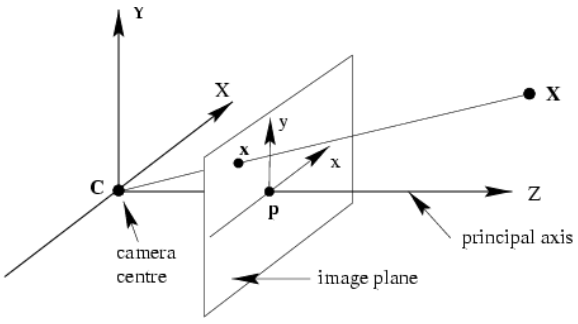


Ilustración 8 Elementos de cámara

Para proyectar los objetos del mundo real en el plano de imagen, se ven involucrados tres sistemas de coordenadas diferentes; el sistema de coordenadas del mundo (3D), el propio de la cámara (3D) y el sistema de coordenadas del plano (2D), *Ilustración 8*.

La matriz de proyección de la cámara o matriz de extrínsecos [$\boldsymbol{R} \mid \vec{\boldsymbol{t}}$], representa la relación entre el sistema de coordenadas de la cámara con el del mundo. La matriz define como rota (matriz 3x3, R) y se traslada (vector 3x1, $\vec{\boldsymbol{t}}$) el sistema de referencia de la cámara respecto al sistema del mundo. Esta matriz es el objetivo último de cualquier aplicación de *tracking,* el resultado a obtener. Se ha de calcular para cada frame de entrada, es decir, que varía continuamente con cada movimiento del usuario.

Por otro lado, se define la matriz de cámara o matriz de intrínsecos, la cual contiene los parámetros principales de la cámara calibrados, como puede ser el FOV ("*Field of View*") u otros términos asociados a la distorsión de la lente. Esta matriz, a diferencia de la matriz de extrínsecos, se calcula en un proceso de calibración previo y se mantiene constante, de no ser que se cambien los parámetros de proyección de la cámara.

Para proyectar la escena en el plano de imagen se define la Ecuación 2.1.1.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2.1.1)$$

Donde se multiplica cada punto 3D [X, Y, Z, 1] por las matrices de extrínsecos [$\boldsymbol{R} \mid \vec{\boldsymbol{t}}$], donde los valores r_{xy} indican las rotaciones que sufren los ejes de la cámara y los valores t_x indican la traslación del centro óptico respecto del origen de coordenadas de la escena, y por los intrínsecos (f_x, f_y, c_x, c_y), que representan los parámetros internos de proyección de la cámara, lo que dará un vector 2D (u, v, 1) que serán las coordenadas 2D del punto en la imagen. El valor s es un factor de escala para homogeneizar el resultado.

2.2 SfM (Structure from Motion)

SfM es una técnica para la obtención de puntos 3D del entorno real partiendo de sus proyecciones en imágenes (puntos 2D). Como resultado de la aplicación de esta técnica se obtiene una reconstrucción 3D del entorno que puede ser empleada más tarde para realizar un proceso de *tracking*.

En esta parte de reconstrucción 3D mediante *SfM*, se introducen varias imágenes desde perspectivas diferentes del mismo entorno, denominadas *keyFrames*. Para cada imagen se detectan los *features* (los puntos más característicos de la imagen). A continuación se deben identificar a aquellos que representan el mismo punto en los diferentes *keyFrames*, obteniendo de este modo la posición de un mismo *feature* en los diferentes *keyFrames*. Con dicha información se realiza una triangulación que determina la posición 3D de cada punto en el espacio real. De forma que se consigue una nube de puntos 3D (*Ilustración 9*).

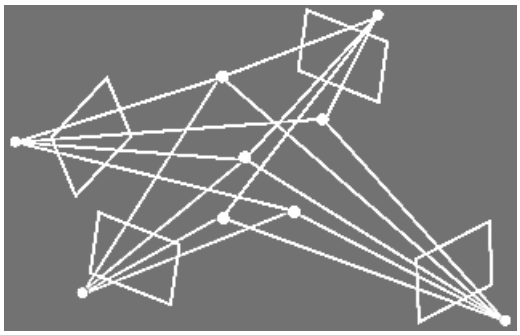


Ilustración 9 Triangulación 2D-3D

De manera más concreta, para determinar la posición 3D de un punto perteneciente a la escena, los métodos clásicos de fotogrametría necesitan conocer la posición 3D y orientación de la cámara o la posición 3D de una serie de puntos de control. Utilizando la primera, en ausencia de GPS y brújula, se puede utilizar la triangulación para reconstruir la geometría de la escena, mientras que con la segunda, se han de identificar manualmente los puntos de control dentro de las imágenes de entrada.

Por el contrario, la técnica de *SfM* no requiere conocer esa información para reconstruir la escena. La posición de la cámara y la geometría son reconstruidas simultáneamente mediante la identificación de *matching features* en las múltiples imágenes. Estos *features* son localizados en las diferentes imágenes, permitiendo realizar unas estimaciones iniciales sobre la posición de la cámara y las coordenadas del objeto, las cuales son refinadas después utilizando un método de minimización no lineal de mínimos cuadrados. A diferencia de la fotogrametría tradicional, la posición de la cámara obtenida mediante *SfM* carece de escala y orientación respecto a las coordenadas de la escena. Consecuentemente, la nube de puntos 3D se genera en un espacio de coordenadas de imagen relativo, que debe ser alineado con el sistema de coordenadas del entorno real.

En la mayoría de los casos, la transformación del sistema de coordenadas obtenido de *SfM* al sistema de coordenadas de la escena puede ser hallado utilizando una transformación 3D de similitud basado en un pequeño número de puntos de control conocidos, cuyas coordenadas están referenciadas respecto al sistema de coordenadas de la escena.
En resumen, en la Ilustración 10 se puede observar el algoritmo de *SfM* y los procesos que lo componen.

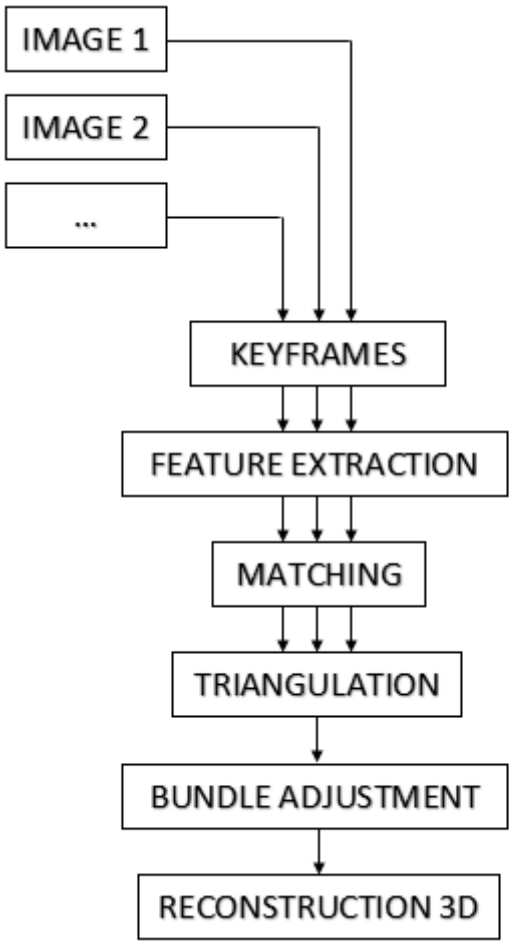


Ilustración 10 Flujo SfM

En los siguientes apartados se describe detalladamente cada uno de los procesos que componen el algoritmo de *SfM*.

2.2.1 Extracción de features

La extracción de *features* requiere, en primer lugar, detectar esos *features* dentro de la imagen. No existe una definición universal o exacta de lo que es un *feature*. Un *feature* se puede definir como una parte "interesante" de la imagen. Con esta condición, se pueden clasificar características relevantes de la imagen en los siguientes tipos; bordes, esquinas o puntos característicos.

En *SFM* los más comúnmente usados son los puntos característicos o esquinas. Una esquina se puede definir como la intersección de dos líneas. Una esquina puede ser definida también cómo un punto para el cual existen dos direcciones de bordes predominantes alrededor suyo. Por otro lado, un punto característico es un punto en la imagen que tiene una posición definida y puede ser localizado de forma robusta. Esto significa, que un punto puede ser una esquina o bien, un punto brillante en un fondo oscuro.

Un buen algoritmo de detección de esquinas/puntos característicos debe contemplar los siguientes criterios:

· Debe detectar todas las esquinas de la imagen.

· No debe detectar esquinas falsas

· Deben estar bien localizadas y el detector ha de ser robusto y eficiente.

Para ser robusto y eficiente, el detector debe ser capaz de detectar el mismo punto en diferentes imágenes obviando las rotaciones, cambios de escala, cambios de iluminación y situaciones con diferentes niveles de ruido.

El detector de esquinas "Harris" [6] es uno de los algoritmos de detección de esquina/puntos más extensamente conocidos. Los puntos detectados por este procedimiento se mantienen invariantes a cambios de escala, rotación, iluminación y ruido.

Una esquina se define en una imagen como una región en la que existen cambios de intensidad en direcciones diferentes. Para detectar esquinas, se define una pequeña ventana alrededor de cada pixel. Esta ventana cubre el pixel situado en $(x, y)^T$. Para la ventana se define una matriz **D**. Los elementos de esta matriz 2x2 representan los cambios de iluminación dentro del vecindario del pixel. Los cambios de intensidad que tienen lugar en cada pixel indican si hay un borde o no. Los eigen-valores (λ_1, λ_2) de la matriz **D** y el valor de la función de auto-correlación (**R**) de la Ecuación 2.2.1.1 donde r es una constante aplicada a esos eigen-valores, indican los cambios de intensidad en ese pixel.

$$R = \lambda_1\lambda_2 - r\,(\lambda_1 + \lambda_2) \qquad (2.2.1.1)$$

Hay tres posibilidades dependiendo del valor obtenido de la auto-correlación (**R**). Si no hay cambios en alguna dirección, es región denominada *flat*. Sin embargo, si hay cambios en la dirección de un borde, entonces la región representa un borde. Por último, si hay cambios significativos en todas las direcciones, la región representa una esquina (*Ilustración 11)*.

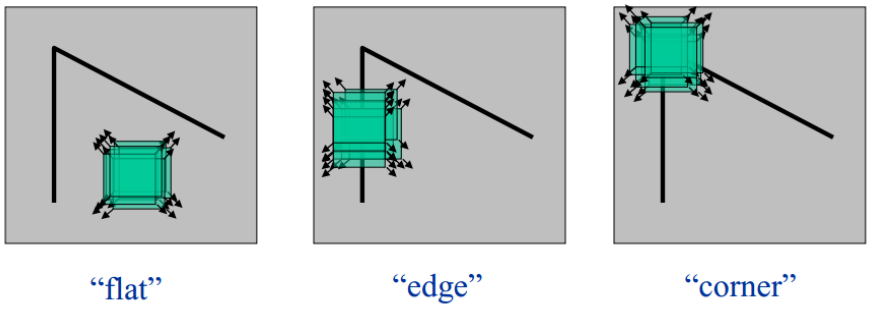


Ilustración 11 Método esquinas "Harris"

Cuando se utilizan los *features* en aplicaciones en tiempo real con un alto *frame-rate*, se necesita un detector de *features* de alta velocidad. El método de Harris [6] es bueno para aplicaciones con *features* de alto rendimiento, no obstante, son métodos con un elevado coste computacional, haciéndolos no deseables para aplicaciones en tiempo real. En este contexto, el detector de esquinas FAST [7] se presenta como un candidato idóneo para la tarea.

[7] funciona considerando un círculo de dieciséis pixeles alrededor del candidato a esquina I_p, como se observa en la *Ilustración 12*. El detector clasifica p como una esquina si existe un set N de pixeles contiguos en el círculo que sean todos más brillantes que la intensidad del pixel candidato I_p más un umbral t. El pixel p también se clasifica como esquina si todos los pixeles contiguos en el círculo son más oscuros que $I_p - t$.

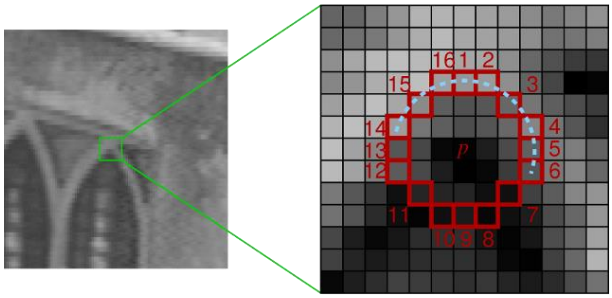


Ilustración 12 Detector de features FAST

2.2.2 Matching

La técnica de *SfM* debe asociar los nuevos *features* con los ya obtenidos en los *keyFrames* para triangular la posición de la cámara y la de los propios puntos. La asociación de *features* se utiliza para identificar la posición de un mismo punto en los diferentes *keyFrames*. Es por ello que, una vez que el proceso de detección de *features* está hecho, se ha de realizar el emparejamiento de *features*. A este proceso de emparejamiento se le llama *matching*.

Para emparejar los puntos coincidentes entre los *keyFrames*, se utilizan los descriptores de los puntos. Los descriptores son vectores que almacenan la información de intensidad de los pixeles alrededor de un *feature*. Por tanto, los puntos que tienen el descriptor más similar son considerados *matches*.

Otra cuestión a tener en cuenta en el proceso de *matching* es que parte de los *matches* obtenidos son falsos. Si estos *matches* erróneos no son eliminados, la solución

(reconstrucción y *tracking*) serán erróneos también. Por esta razón, las técnicas de *matching* deben ser robustas.

Al realizar el proceso de detección de *features*, se utilizan descriptores. Para ello, es necesario identificar el vecino más próximo al punto. Los descriptores de los puntos se comparan utilizando la distancia. La distancia mínima indica el vecino más próximo y su correspondiente punto emparejado en el proceso de *matching*.

Por otro lado, también se utiliza la correlación cruzada normalizada en el proceso de *matching* con parches planos alrededor del punto de interés.

En general, el uso de descriptores convencionales tiene un coste computacional muy alto para aplicaciones de AR. Hoy en día, el desarrollo de algoritmos de visión para teléfonos inteligentes y dispositivos embebidos con poca memoria y baja capacidad de cómputo ha cambiado las condiciones para este tipo de aplicaciones. Por esta razón, el reto es utilizar descriptores más fáciles de calcular y con menor uso de memoria que se mantengan robustos ante cambios de escala, rotación, iluminación y ruido.

En este contexto, aparecen los descriptores FREAK [8]. Estos descriptores se basan en el sistema de visión humano y especialmente en su retina. Por esta razón se denomina al método *Fast Retina Keypoint* (FREAK). En él, se calcula eficientemente una cascada de cadenas binarias comparando pares de intensidades de imágenes utilizando un patrón retinal de muestras. Curiosamente, seleccionar pares de imágenes para minimizar la dimensionalidad del descriptor genera un patrón altamente estructurado que imita la búsqueda que realizan los ojos humanos.

El patrón de muestreo de la retina utiliza la rejilla retinal de muestreo, la cual consiste en círculo con una alta densidad de puntos cerca del centro. Cada punto de muestreo necesita ser suavizado para resultar menos sensible al ruido. La *Ilustración 13* muestra el patrón utilizado para generar los descriptores.

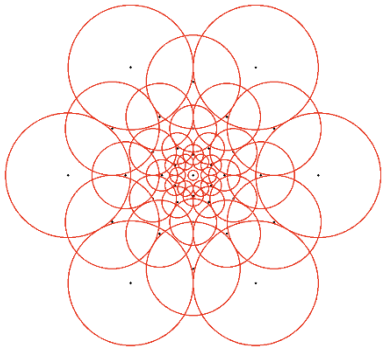


Ilustración 13 Patrón FREAK

2.2.3 Triangulación

Tras los procesos de extracción y *matching*, se realiza el proceso de reconstrucción 3D del entorno mediante esos *features*. Esto significa que se debe encontrar el valor de los tres ejes (X, Y, Z) para los *features* detectados. Como resultado de la reconstrucción, se obtiene un conjunto de posiciones 3D. A esto se le denomina *Point Cloud*.

Las imágenes representan la realidad en un espacio 2D. La reconstrucción de un objeto real se puede generar a partir de diferentes imágenes 2D del objeto con diferentes

perspectivas. Con este propósito, la tercera dimensión de cada punto se haya mediante un proceso de triangulación.

La *Ilustración 14* muestra la geometría epipolar [10] de un par de cámaras *pin-hole* estéreo. Un punto ***P*** en el espacio tridimensional se proyecta en un el plano de imagen en perspectiva en el punto de intersección entre la recta, formada por el punto y el centro óptico de la cámara, con el plano (O_R y O_L), resultando en los puntos p_R y p_L. Si se proporcionan estos puntos y la geometría de ambas cámaras es conocida, se pueden determinar las dos líneas. Estas líneas deberán intersecarse en el punto ***P***. Utilizando algebra lineal básica, se puede calcular la intersección de manera sencilla.

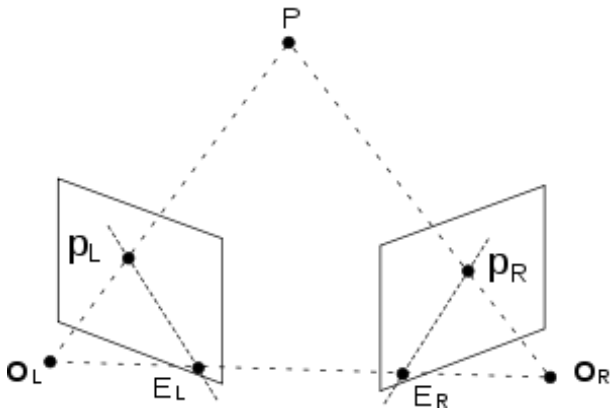


Ilustración 14 Geometría epipolar

Mediante este proceso se obtiene el *Point Cloud* que representa el mapa 3D del entorno. Además, se obtiene la matriz necesaria para conseguir la matriz $[\boldsymbol{R} \mid \vec{\boldsymbol{t}}]$.

2.2.3 Bundle adjustment

Bundle Adjustment (BA a partir de ahora) [9] es el problema de refinar una reconstrucción 3D para generar una estructura conjunta óptima y estimar los parámetros de las cámaras. Se trata en realidad de un problema de estimación de parámetros geométricos. Los parámetros son una combinación de coordenadas 3D de los *features* con la calibración de la posición de las cámaras.

BA es casi invariablemente el último paso en cada método de reconstrucción basado en multi-perspectivas de *features* utilizado para obtener la estructura 3D y la estimación de los parámetros de movimiento. Mediante las estimaciones iniciales, BA simultáneamente refina la estructura y la posición de las cámaras utilizando una minimización del error de reproyección (Ecuación 2.2.3.1) de los puntos estimados respecto a los observados.

$$\sum_{\boldsymbol{i}} \mathrm{d}\,(x_i, X_i)^2 + d\,(y_i, Y_i)^2 \qquad (2.2.3.1)$$

2.3 Tracking

Las técnicas de *tracking* o *motion tracking* consisten en calcular la posición y orientación de la cámara a partir de las imágenes que captura, utilizando elementos del entorno que son proyectados en la imagen para triangular su posición con respecto a ellos. Como se ha explicado en el capítulo 1, existen diferentes técnicas de *tracking*. Si bien los basados en marcadores son muy populares en aplicaciones móviles, el presente TFG se centra en las técnicas de *tracking* sin marcadores o ningún modelo conocido.

Para poder realizar el proceso de *tracking* sin tener ninguna referencia previa del entorno, es necesario realizar una reconstrucción del mismo. Para ello se emplean técnicas como la citada en el apartado 2.2. Mediante esa técnica de *SfM* se obtiene una nube de puntos 3D que representa al entorno.

El mecanismo de funcionamiento del *tracking* consiste en analizar cada *frame* de video capturado por la cámara del dispositivo en busca de *features*. Para ello, mediante los parámetros de cámara del *frame* anterior se proyectan los puntos 3D de la nube de puntos en el *frame* actual.

El objetivo del *tracking* consiste en emparejar estos *features* 2D con los puntos 3D de la nube de puntos. Para ello existen varias técnicas, una muy utilizada consiste en tomar un parche de tamaño fijo alrededor de cada feature en el *keyFrame* en el que se ha visto el punto por primera vez. Con este parche se realiza una búsqueda en la región en la que se encuentra la proyección del punto en el *frame* actual y mediante un coeficiente de correlación cruzada se determina el punto cuyo entorno tiene más similitud tiene con el parche mediante el *template matching*.

Las técnicas de *template matching* han sido usadas por multitud de autores con el fin de emparejar *features* 2D con puntos 3D. Los métodos de *tracking* sin marcadores como *visual SLAM* [11], emparejan puntos 3D de la nube de puntos reconstruida con sus correspondientes puntos 2D en las imágenes. Para conseguirlo, se utilizan principalmente la detección de *features* y el *matching*. Los algoritmos *SIFT* [12], *FREAK* y *SURF* [13] son muy robustos gracias a su invariabilidad frente a la iluminación, escala y orientación de la imágenes. En consecuencia, la probabilidad de un emparejamiento correcto aumenta. Sin embargo, los procesos de detección de *features* y *matching* requieren un coste computacional elevado.

Otros autores utilizan técnicas de *optical flow* para estimar la velocidad de imagen o el cambio de un punto específico entre *frames*. Aunque este método tiene un coste computacional menor, pero la estimación del desplazamiento del *feature* no es tan precisa como en otros métodos y además es propenso a la deriva.

Por otro lado, parches relativamente grandes pueden servir como *features*. Un ejemplo de ello se ve en [3]. Donde, para ser capaces de detectar correctamente estos parches, utilizan métodos de *template matching*. Los procesos de *template matching* utilizan diferentes operadores para determinar la similitud entre parches. La "suma de diferencia absoluta" es uno de estos operadores. Sin embargo, para conseguir mejores resultados obviando variaciones, por ejemplo de iluminación, hay otras alternativas, como es el coeficiente de correlación cruzada o el coeficiente de correlación cruzada normalizado.

Los métodos de *template matching* pueden fallar en la obtención de correspondencias debido al movimiento de la cámara y a las deformaciones de perspectiva. La orientación de la superficie puede ser utilizada para tener en cuenta esas deformaciones. Algunos autores sugieren utilizar métodos probabilísticos para estimar la orientación más probable para cada superficie. A su vez, en [3] se realiza una simplificación: se asume que un punto 3D siempre pertenece a una superficie orientada hacia la cámara donde

ha sido visto el punto por primera vez. Asume también que la apariencia del parche no cambia en gran medida.

Sin embargo, [14] obtiene un modelo de reconstrucción muy efectivo para estatuas y elementos arquitectónicos. El trabajo enfatiza el uso de la orientación de las superficies para obtener buenos resultados.

Parte Segunda

DESARROLLO

Algoritmo “WaPT” Capítulo 3

En este apartado se profundizará en el funcionamiento del algoritmo “WaPT”. Este TFG se basa en la implementación de este algoritmo.

Como se ha comentado en el apartado 1.1, hoy en día los métodos de *tracking* óptico son utilizados en muchos campos como el guiado de robots o en AR. En esencia, estos métodos establecen una relación entre la representación interna de la escena real (nube de puntos 3D) y las imágenes que son capturadas por la cámara del dispositivo. En el caso de *tracking* sin marcadores, la solución más común consiste en emparejar puntos 3D con sus correspondientes puntos 2D en la imagen.

Entre estas técnicas es común hacer uso de los puntos característicos para el emparejamiento. De esta forma se consigue que el método sea robusto en condiciones deficientes de iluminación, orientación y cambios de escala. Aun así, presentan algunos problemas de eficiencia. Por esta razón, los puntos característicos son utilizados solamente por el método para el emparejamiento inicial y otras técnicas, como el *optical flow*, se utilizan para localizar las correspondencias de los puntos 2D entre *frames* del vídeo.

La técnica del *optical flow* estima de manera iterativa el desplazamiento de cada punto característico de *frame* en *frame*. Esta técnica tiene el problema de que el error es acumulativo (en inglés *drift*), debido a las iteraciones. En consecuencia, para obtener un resultado más preciso y minimizar el *drift* en el proceso de *tracking*, se utilizan métodos como el *template matching*.

En estos métodos, para cada punto 3D de la nube de puntos, se toma un parche alrededor del punto en las imágenes de referencia y se guarda como un *template* de referencia. Luego, se utilizan técnicas de *template matching* para el *tracking*. Sin embargo, debido al movimiento de la cámara y la geometría de la escena real, en emparejamiento con los *templates* de referencia puede fallar debido a deformaciones de perspectiva, como se puede observar en la *Ilustración 15*. No obstante, con la intención de solucionar el mencionado problema, el *template* de referencia puede ser deformado teniendo en cuenta la posición de la cámara y la geometría de la escena.

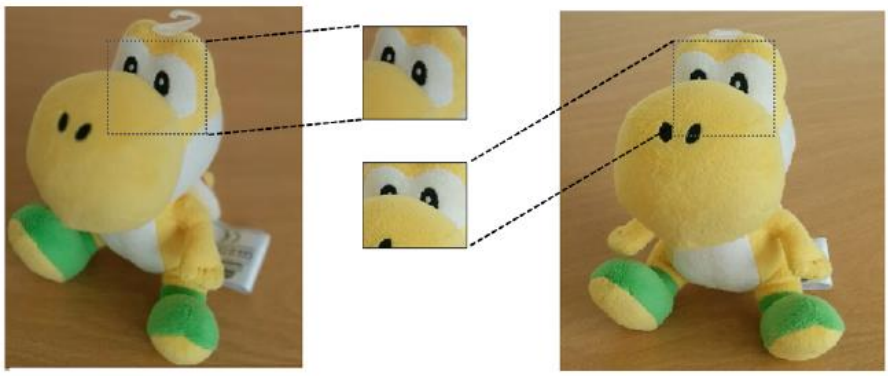


Ilustración 15 Ejemplo de error de paralaje

En este TFG se hace uso de algoritmo denominado "WaPT" [4] (*Warped Template Patch Tracking*) para solucionar ese efecto de deformación en la perspectiva. En él se mejora el proceso de transferencia del parche, generando un parche más preciso. En consecuencia, el *template matching* es más preciso también.

"WaPT" extiende la representación interna de la escena real: cada punto 3D tendrá un vector normal que estimará la orientación de la superficie real en la que está situado. De esta forma, el proceso de reconstrucción no buscará solamente los puntos 3D, sino que estimará también estos vectores normales. "WaPT" utilizará esta orientación de los vectores para realizar el proceso de *template matching*.

Tal y como se describe en el trabajo de [3], los puntos 3D son considerados *features* que son proyectados en las imágenes con el fin de localizar la posición de la cámara en tiempo real. Con el fin de realizar este proceso de forma más precisa, se definen 2 partes diferenciadas:

- Reconstrucción 3D:

En esta parte se realiza la reconstrucción 3D del entorno y la estimación de los vectores normales para cada punto 3D. Esta vector define la orientación de la superficie en la que se encuentra el punto 3D. Este proceso se realiza de manera *offline* y el objetivo principal es conseguir una nube de puntos del entorno y la orientación del vector normal con el fin de realizar en la siguiente parte una transferencia del parche más precisa.

- Tracking:

En esta parte, para cada *frame* de entrada se realizan los siguientes pasos:

1- Utilizando el vector normal calculados en el paso anterior, se deforma el parche de referencia, con dicho fin, se toma un parche de tamaño concreto desde la imagen actual y se transfiere a la imagen de referencia (aquella en la que primero se ha visto el punto durante la fase de reconstrucción). Después, utilizando el parche de referencia deformado, se transfiere a la imagen actual de nuevo, localizando este parche dentro de la imagen. El centro de este parche es tratado como un *feature*.

2- Una vez obtenidos todos los *features y* sus vectores en el paso previo son usados para calcular la nueva posición de la cámara, que se utilizará para proyectar los elementos virtuales de la aplicación y a su vez realizar el mismo proceso en el frame siguiente.

Las secciones siguiente tratan de describir en detalle los dos pasos realizados por el algoritmo "WaPT".

3.1 Reconstrucción 3D (offline)

El objetivo de la parte de reconstrucción 3D (pre-proceso) es la de generar una representación 3D del entorno. Esta representación debería ser lo más precisa posible para realizar de manera correcta la tarea de *tracking*. El archivo de salida que se obtiene de esta fase es una nube de puntos con un vector normal asociado a cada punto, que aproxima lo orientación de la superficie en la que se encuentra ese punto.

La nube de puntos se obtiene usando el método *SfM*. *SfM* combina técnicas multi-perspectiva junto a un proceso de ajuste. Como entrada se utilizan imágenes del entorno. Las imágenes que se utilizan en *SfM* se denominan *keyFrames*. El objetivo de "WaPT" es estimar el vector normal de cada punto obtenido con *SfM*. Para lograr este objetivo, se realiza un proceso de minimización.

Considerando el caso representado en la *Ilustración 17*. Hay dos keyFrames en los que el mismo punto es visible. El keyFrame en que se ha visto primero el punto se denomina Reference keyFrame. El punto tiene asociado un vector normal a la superficie en la que se encuentra. La *Ilustración 18* muestra las proyecciones del punto 3D en cada keyFrame. También muestra los parches alrededor del punto 3D y el plano definido por el punto 3D y su vector normal.

Ilustración 16 Ejemplo reconstrucción 3D

Teniendo en cuenta estas dos definiciones:

Transferred patch: se trata de un parche cuadrado definido alrededor de un punto 3D proyectado en todos los *keyFrames* en los que es visible, excepto en el *reference keyFrame (Ilustración 18)*.

Reference patch: es el parche obtenido cuando un *transferred patch* es transferido al *reference keyFrame*; el parche deformado (*Ilustración 17)*.

El proceso para obtener un *reference patch* es el siguiente; dado un punto 3D, su proyección en el *keyFrame* se utiliza para definir un *transferred patch*. Con ese *transferred patch*, se realiza la proyección inversa en el plano asociado con el punto 3D. Este nuevo polígono es proyectado en el *reference keyFrame* generando el *reference patch*. (*Ilustración 17*).

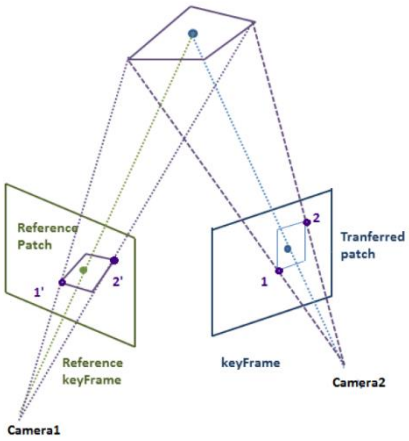


Ilustración 17 Proceso de transferencia de parche

El objetivo del algoritmo de minimización es el de encontrar el vector normal que minimiza las diferencias entre los dos parches.

"WaPT" utiliza el algoritmo de minimización "Levenberg-Marquardt". Este algoritmo consiste en un proceso iterativo donde se elige un vector normal aleatorio como tesis de partida. El plano definido por el vector normal es usado para transferir los *transferred patch* hacia el *reference patch*. Tras esto, el algoritmo evalúa la diferencia entre el *transferred patch* y el *reference patch*. El algoritmo de minimización ajusta el vector normal de forma que la diferencia entre todos los *transferred patch* con el *reference patch* se lo menor posible. El algoritmo sigue iterando hasta que la diferencia con la anterior iteración no supera un umbral establecido.

3.1.1 Plano característico

En el espacio de proyección se define un plano como muestra la ecuación (1). Para definirlo, se necesitan tres puntos ($\overrightarrow{X_1}$, $\overrightarrow{X_2}$, $\overrightarrow{X_3}$) que pertenezcan al plano. $\overrightarrow{X_1}$ se utiliza para representar el punto 3D. Los otros dos puntos ($\overrightarrow{X_2}$ y $\overrightarrow{X_3}$), se calculan utilizando el vector normal y la restricción impuesta por la ecuación (2), donde $(a, b, c)^T$ es el vector normal y $(x, y, z)^T$ el punto 3D en el espacio euclídeo. Hay que tener en cuenta que d está definido por la ecuación (3).Esta ecuación presenta singularidades cuando el vector normal está alineado con alguno de los ejes principales.

$$\vec{\pi} = \begin{pmatrix} \left(\overrightarrow{\hat{X}_1} - \overrightarrow{\hat{X}_2}\right) x \left(\overrightarrow{\hat{X}_2} - \overrightarrow{\hat{X}_3}\right) \\ -\overrightarrow{\hat{X}_3}^{\vec{T}} \left(\overrightarrow{\hat{X}_1} \, x \, \overrightarrow{\hat{X}_2}\right) \end{pmatrix} \qquad (1)$$

$$ax + by + cz + d = 0 \qquad (2)$$

$$d = -(ax_1 + by_1 + cz) \qquad (3)$$

3.1.2 Transferir el parche

Cada pixel del *transferred patch* es reproyectado en el plano, es decir, cada pixel del *transferred patch* define un punto que representa la intersección entre su rayo de reproyección y el plano. Después, estos puntos son proyectados sobre el *reference keyFrame*, como se puede ven en la *Ilustración 17*.

La reproyección del punto en la imagen está definida en la ecuación (4), donde P^+es la matriz de proyección pseudo-inversa "Moore-Penrose" y $\vec{C}$ es el centro de la cámara en el sistema de referencia global. Además. Un punto que pertenezca al plano debe cumplir la ecuación (5). Resolviendo este sistema de ecuaciones, se obtienen los puntos. Tras esto, las proyecciones de esos puntos en el *reference keyFrame* ya están conseguidas

$$\vec{X} = P^+ * \vec{x} + \lambda * \vec{C} \qquad (4)$$

$$\pi^T * \vec{X} = 0 \qquad (5)$$

3.1.3 Evaluar la diferencia entre parches

Para evaluar la diferencia entre las imágenes del *transferred patch* y el *reference patch* se utiliza el coeficiente de correlación cruzada. El coeficiente de correlación cruzada mide la similitud entre dos imágenes, donde un emparejamiento perfecto es un 1 y -1 en caso contrario. "WaPT" trata de conseguir la máxima similitud entre ambos. Así pues, el valor del coeficiente de correlación ha de ser tan alto como sea posible. Sin embargo, este valor es utilizado en un proceso de minimización. Por esa razón, para evaluar la diferencia entre parches, se utiliza la función objetiva mostrada en la ecuación (6), siendo la correlación cruzada el coeficiente de correlación cruzada normalizado del parche seleccionado en el *frame i*, y siendo α y β los dos ángulos utilizados en la parametrización del vector normal.

$$\min(\alpha, \beta) \sum_i (1 - correlaciónCruzada(\alpha, \beta)) \qquad (6)$$

3.2 Tracking (online)

Como en todos los procesos de *tracking*, el objetivo principal es obtener la matriz de proyección de la cámara para cada *frame* de entrada. En este caso, los datos de entrada necesarios para hacerlo son:

1. Set de imágenes de referencia: se han de introducir las imágenes de referencia junto con sus matrices de proyección. Estas imágenes son los *keyFrames*. Hay que recordar que el *reference keyFrame* es el *keyFrame* en el que el punto 3D se ha visto por primera vez en la reconstrucción 3D.

2. Nube de puntos: un archivo de texto con la posición de todos los puntos 3D junto a sus vectores normales. Este archivo de obtiene de la fase de reconstrucción. Cada punto 3D almacena la información de su *reference keyFrame* dentro del conjunto de imágenes de referencia.

El objetivo es localizar las proyecciones de los puntos 3D en el *frame* actual con el fin de localizar la imagen en el entorno. Con este propósito, *n* puntos de la nube de puntos son elegidos como *features*, de forma uniformemente distribuida dentro la imagen. Tras varios procesos empíricos, se ha decidido elegir *n*=50 puntos para los experimentos realizados en este TFG. Estos puntos son proyectados en el *frame* actual con la matriz de proyección del *frame* anterior. Se supone que el movimiento de la cámara entre *frames* es pequeño, es decir, que la matriz de proyección no varía mucho de un *frame* a otro y por tanto los puntos se han movido poco.

Para obtener la posición correcta de los puntos 3D en la imagen actual, se ha de realizar un ajuste. Este ajuste viene causado por el hecho de proyectar los puntos con la matriz de proyección del *frame* anterior (*Ilustración 18*). Como se comenta arriba, se supone que la cámara se ha movido respecto a la posición anterior, por ello las proyecciones de los puntos 3D corresponden al *frame* anterior y se han de reposicionar en el *frame* actual. De forma que para cada proyección aproximada de cada punto 3D se define un parche a su alrededor. Tras esto, se realizan 2 pasos para realizar el ajuste de forma correcta:

1. Se transfiere el parche desde la imagen actual al *reference keyFrame* del punto 3D. De esta forma, se obtiene el *reference patch*.

2. Se realiza un proceso de búsqueda, es decir, se busca el parche con mayor similitud dentro de *frame* actual.

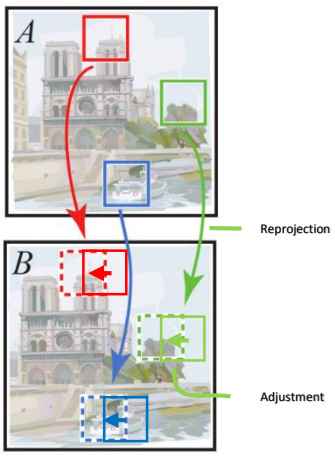


Ilustración 18 Ajuste "WaPT"

3.2.1 Transferencia del parche

El parche se transfiere utilizando el mismo proceso que en la fase de reconstrucción 3D, apartado 3.1. Primero, se toma la proyección del punto 3D en el *frame* actual y se utiliza como centro para el *transferred patch*. Utilizando el vector normal del punto 3D, se obtiene el plano. Entonces, usando este plano, se reproyecta cada pixel en cada uno de los *transferred patch*. La intensidad de cada pixel en el *reference patch* se obtiene utilizando una interpolación bilineal de intensidad. De esta forma, se obtiene la imagen del *reference patch*.

3.2.2 Ajuste de la proyección del punto

El objetivo de este paso es localizar el parche en el *frame* actual con la máxima similitud al *reference patch*. Para ello, se utiliza el coeficiente de correlación cruzada dentro del área de búsqueda. El área de búsqueda es una ventana de tamaño fijo W dentro del *frame* actual cuyo centro es la proyección del punto 3D. Experimentalmente, se ha comprobado que utilizando un área de búsqueda de 128x128 pixeles en imágenes de tamaño 640x480 se obtienen buenos resultados.

Para conseguir más precisión en el proceso de búsqueda, se utiliza una reducción piramidal de tres niveles de la siguiente manera:

· De la reducción de tres niveles de W se obtienen W_{L0}, W_{L1} y W_{L2}, donde $W = W_{L0}$.

· I es la imagen original, es decir, el *frame* actual.

· Además, se calcula la misma reducción piramidal para el *reference patch* R, obteniendo R_{L0}, R_{L1} y R_{L2}, donde $R = R_{L0}$.

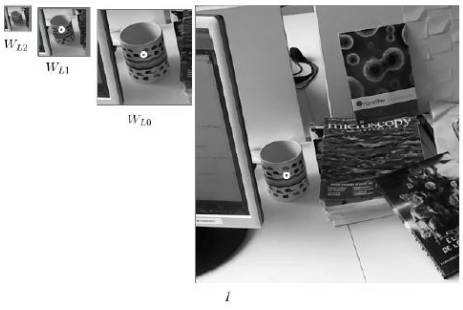


Ilustración 19 Niveles Piramidales

R_{L0}, R_{L1} y R_{L2} son la misma imagen con diferentes resoluciones. Para resultar más robusto frente al ruido, los parches utilizados en el proceso de búsqueda son regiones con tamaño fijo situados en los centros de R_{L0}, R_{L1} y R_{L2}. Nuevamente, mediante procesos empíricos, se ha determinado utilizar regiones de 8x8 pixeles.

Una vez que los niveles piramidales se han definido, comienza un proceso iterativo. En este proceso iterativo, el algoritmo utiliza en primer lugar el nivel piramidal más bajo (W_{L2} y R_{L2}). Después el resultado se propaga hacia los niveles superiores (*Ilustración 19*).

Así, primero se buscará R_{L2} dentro de W_{L2} utilizando el coeficiente de correlación cruzada, centrando la región de 8x8 pixeles en el centro de R_{L2}. La función devuelve la posición dentro de W_{L2} para la región con más similitud con R_{L2}. Esta posición se propaga al

siguiente nivel (W_{L1}). El área de búsqueda de W_{L1} se redefine teniendo en cuenta la posición calculada en el paso previo, obteniendo $W_{L1'}$. El proceso se repite localizando R_{L1} dentro de $W_{L1'}$.

Este proceso iterativo se repite hasta que la posición se propaga al nivel final (W_{L0}) y finalmente a la imagen (I). En la *Ilustración 19* se puede ver la propagación entre los diferentes niveles.

Este proceso se realiza para los *n* puntos, obteniendo un conjunto de *n features* que serán utilizados para calcular la matriz de proyección de la cámara en el *frame* actual.

Planteamiento general

Como se ha mencionado en apartados anteriores, el objetivo de las aplicaciones de AR es la obtención de la localización en tiempo real del usuario, o lo que es lo mismo, del dispositivo. Teniendo en cuenta este objetivo, las técnicas empleadas para realizar dicho *tracking* van evolucionando en la medida en la que el modo de uso de las aplicaciones evoluciona.

En el capítulo 3, se ha definido el algoritmo WaPT. Éste, tiene como objetivo mejorar la precisión de los algoritmos de *tracking* por visión artificial, haciendo uso de la información que aporta la curvatura predominante de cada uno de los puntos del espacio real. Sin embargo, el cálculo de dicha curvatura para cada uno de los puntos supone un coste computacional añadido al proceso. Es por ello que salen a relucir ciertas cuestiones como ¿mejora el proceso de localización teniendo en cuenta la curvatura predominante de cada punto? , ¿Cuánto mejora?, ¿Cuán preciso debe ser este cálculo para que algoritmo sea eficiente? , ¿Cómo varía el uso de dicho vector en el proceso de localización ante diferentes geometrías?, etc.

Es por ello, que en este TFG se realizarán los trabajos y desarrollos pertinentes con el fin de determinar con precisión el comportamiento del algoritmo WaPT y de las técnicas de *template matching*.

En este capítulo se presenta la idea global que el sistema ha de desarrollar con el fin de realizar el mencionado análisis.

Diseño

En este apartado se presenta el diseño global del sistema de análisis desarrollado en el presente TFG. En el capítulo 3 se ha especificado la implementación del algoritmo WaPT, del que se hará uso en este TFG con el fin de analizar el comportamiento de las técnicas de *template matching* teniendo en cuenta el cálculo de la localización en función de la deformación aplicada a las imágenes.

En la *Ilustración 20* se puede observar el diseño de la arquitectura del algoritmo WaPT. Como se puede apreciar en la mencionada ilustración, y como se ha explicado en el capítulo 3, este algoritmo consta de dos grandes módulos; la reconstrucción 3D y el *tracking*. En el caso que nos ocupa, el resultado del módulo de reconstrucción 3D no vendrá dado por la aplicación de la técnica de *SfM* si no que será conocido de antemano.

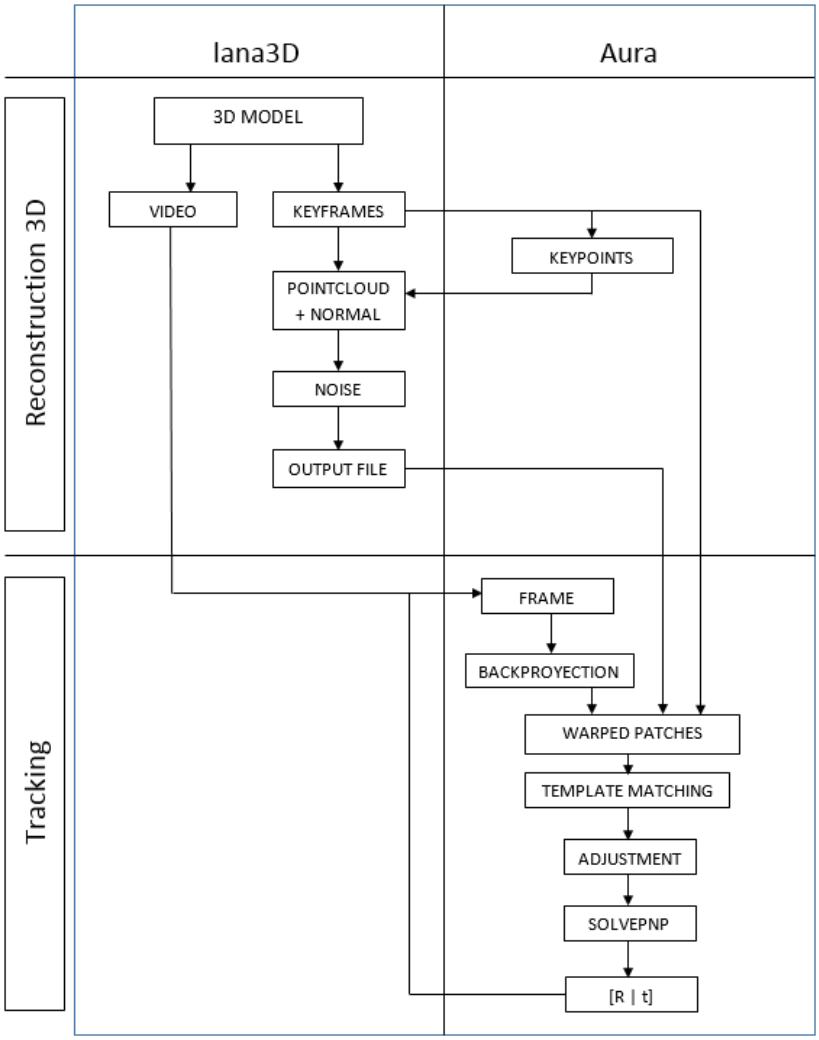


Ilustración 20 Flujo de desarrollo

En el diagrama se observa el flujo del algoritmo de principio a fin. Como se ha comentado, consta de 2 partes diferenciadas, la reconstrucción 3D y el *tracking.* La primera parte se realiza principalmente con el motor gráfico de lana3d. En ella, primero se obtienen las imágenes (*keyFrames*) del entorno, en este caso se crea el modelo 3D sintético, del que se extraen 2 conjuntos de archivos. El conjunto de *frames* que conforman el video a analizar, que se utilizará más tarde en el proceso de *tracking*, y el

conjunto de *keyFrames* que se utilizan en esta parte para conseguir la reconstrucción 3D del entorno.

Mediante estos *keyFrames* se triangula la posición de los diferentes puntos 3D, por lo que se necesita pasar los *keyFrames* por Aura para detectar *features* en las imágenes. Mediante esos *features*, en el caso que nos ocupa, eligiendo doscientos de ellos, se triangula la posición de los puntos 3D y se estima la orientación de sus vectores normales, obteniendo la nube de puntos 3D.

Tras esto, hay una fase opcional de adición de ruido a los vectores normales y de ahí, por último, se obtiene el archivo de reconstrucción que contiene la posición de la cámara para los distintos *keyFrames*, así como la posición 3D de los puntos y sus normales, y las proyecciones de esos puntos dentro de cada *keyFrame*.

En la segunda parte de algoritmo se realiza el proceso de *tracking*, por el cual se calcula la posición de la cámara dentro de cada uno de los *frames* que conforma el vídeo. Esta fase es cíclica, de forma que se toma la posición de cámara del *frame* anterior para proyectar los puntos 3D dentro de la imagen. Tras ello, se pasa a deformar los parches alrededor de los puntos en los *keyFrames* y se realiza una búsqueda con estos parches alrededor de la posición del punto dentro del *frame* actual, con lo que se consigue refinar la posición mediante la técnica de *Template Matching*. Tras realizar el ajuste de la posición de los puntos se pasan por un filtro para eliminar los mal calculados y que de esta forma no influyan en el resultado final. Por último, con los puntos que han pasado el filtro, se calcula la nueva posición de cámara, que será usada en la siguiente iteración para repetir el proceso.

Tareas Capítulo 4

Con el fin de realizar el exhaustivo análisis del comportamiento se han llevado a cabo determinadas tareas consistentes en la creación de un modelo 3D sobre el que realizar las pruebas de forma que todos los parámetros sean perfectamente controlables. Mediante este modelo, se han variado los parámetros de interés para comprobar cómo se comporta el algoritmo en esas situaciones. En los siguientes apartados se detallan las principales tareas técnicas llevadas a cabo.

4.1 Creación de modelo sintético

Con el fin de realizar las pruebas de la manera más exhaustiva posible, y comprobar con total veracidad la implicación del cálculo del vector normal en el proceso de *tracking*, es imprescindible crear un entorno totalmente controlado de manera sintética que simula la salida del proceso de reconstrucción 3D. Es por ello, que el objetivo principal de esta tarea es la generación del modelo sintético.

Para la creación de los entornos controlados para las pruebas se ha desarrollado haciendo uso de la librería lana3d. Éste es un motor gráfico de software creado por Vicomtech-IK4.

lana3d es un software creado en el departamento de computación gráfica e interacción de Vicomtech-IK4 con la intención de disponer de un software para la creación de entornos 3D, animaciones y *renderización* de forma que no se dependa de programas comerciales externos. En el programa se permite la creación de escenas 3D en la que se pueden añadir diferentes elementos como cámaras, luces u objetos y posteriormente crear imágenes o videos de esa escena con una iluminación realista.

Ilustración 21 lana3D

Al ejecutar lana3D, en primera instancia para crear un nuevo entorno sintético 3D, se precisa crear una escena sobre la que situar los elementos 3D. Para ello, dentro de la librería, se ha implementado un gestor de creación de nodos. Los nodos son todos aquellos elementos que componen un entorno 3D.

Para este trabajo, los elementos introducidos han sido un nodo "*scene*" (la escena 3D) en el que se define un sistema de coordenadas para la escena. En este nodo "*scene*" se han posicionado el resto de elementos 3D. En este caso han sido un nodo "*ambience light*" (iluminación ambiente), un nodo "*perspective camera*" (una cámara) al que se le han definido los parámetros internos de proyección y un nodo "*mesh*" (un objeto 3D), en este caso el plano o la esfera al que se le ha añadido una textura y se ha situado su centro en la posición (0, 0, 0).

Ilustración 22 Textura

Para el caso del plano, sea creado directamente en el software. Sin embargo para el caso de la esfera, ha sido importada desde un archivo externo procedente de un software comercial (3D Studio Max). Ya que tras crear la esfera con lana3d, se observó que la geometría contenía errores ya que muchas de las caras se repetían o superponían. Así, se decidió utilizar un software externo para la creación de la esfera e importar el modelo en lana3d.

Ilustración 23 Plano en lana3D

Ilustración 24 Esfera en lana3D

El tamaño del plano es de 25’6x16 Uds. (Ancho x Alto) y el radio de la esfera es 7’5 Uds. y se han situado ambos en el origen de coordenadas de la escena. La cámara se ha situado a una distancia de 40 Uds. del origen en el eje Z para las imágenes de referencia y a partir de ahí se ha rotado y trasladado para obtener las imágenes del vídeo.

En un principio, tras una primera batería de pruebas con el plano, se observaron las limitaciones que el experimento contenía asociadas a la geometría del plano. Todos los vectores normales son iguales y además todos los puntos son coplanarios. Por ello, se planteó la idea de realizar otra batería de pruebas con algún objeto de geometría irregular, donde los vectores normales no fuesen todos iguales. Las complicaciones que suponía crear ese objeto llevaron a emplear una forma esférica.

Para aplicar la misma textura (Ilustración 22) que en el plano, la imagen se deforma para adaptarla a la geometría de la esfera, para ello se ajusta la franja central de la imagen a modo de cinturón alrededor de la esfera y se constriñe la imagen en los polos de la esfera, como si se tratara del envoltorio de un caramelo.

4.2 Extracción de keypoints

Una vez obtenida la imagen de referencia para la esfera o la imagen de la textura para el plano, se introducen en el algoritmo de WaPT como imágenes de entrada.

El algoritmo forma parte de una librería de código más grande llamada Aura. Aura se creó en Vicomtech-IK4 con el mismo propósito que lana3d y es el de poseer una herramienta software propia a medida para no depender de otras comerciales externas. En Aura, se implementan una serie de funcionalidades en torno a la AR que permiten crear aplicaciones en este campo. Como se comenta en la introducción del TFG, permite crear aplicaciones de AR con y sin marcadores. En el caso de marcadores además, permite realizarlo con varios de ellos, independizando escenas. De esta forma, aunando Aura y lana3d, se pueden crear aplicaciones muy potentes de AR.

En la primera fase de análisis del algoritmo, se realiza la detección de puntos. Aquí, para cada imagen del video de entrada, se localizan puntos característicos y se almacena la información de su posición en un archivo de texto. En este caso, se introducen las imágenes de referencia como imágenes de entrada para detectar puntos en ellas.

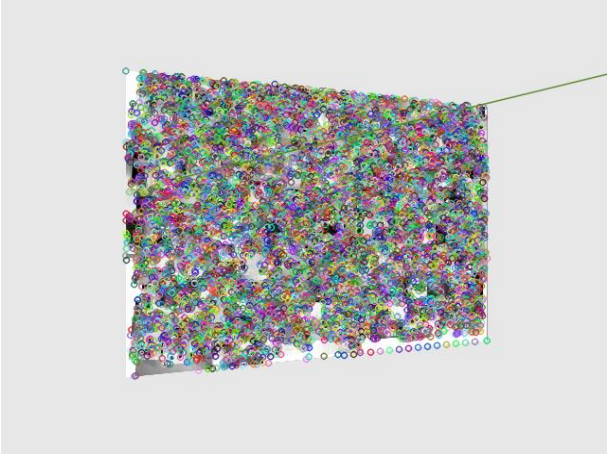


Ilustración 25 featured points para el plano

En la imagen se muestran con círculos los puntos característicos detectados.

El flujo de esta parte del algoritmo es la siguiente. Se convierte la imagen de entrada a escala de grises, tras lo que se buscan puntos con un alto contraste con respecto a sus vecinos. La información que se almacena el archivo de texto es la cantidad de puntos detectados, más de 11000 y 3500 puntos para el plano y la esfera respectivamente, y una lista con un identificador y las coordenadas X e Y de cada punto.

Los puntos están se encuentran ordenados por la coordenada Y, es decir, la imagen se analiza de arriba hacia abajo.

Dado que la cantidad de puntos extraídos de las imágenes es ingente, se ha decidido seleccionar doscientos de esos puntos de forma distribuida. Para ello se ha dividido la cantidad de puntos entre doscientos y se han cogido puntos con ese intervalo.

$$\frac{11.265}{200} \approx 56 \qquad \frac{3.635}{200} \approx 18$$

Uno de cada cincuenta y seis para el plano y uno de cada dieciocho para la esfera.

Ilustración 26 Proyecciones (verde) de los 200 puntos seleccionados para el plano

Ilustración 27 Proyecciones (verde) de los 200 puntos seleccionados para la esfera

4.3 Raycaster

Una vez obtenidos los puntos característicos dentro de las imágenes de referencia es necesario traducir esa información al entorno 3D para obtener la posición y el vector normal de cada uno de los puntos.

Para el plano resulta sencillo traducir las proyecciones obtenidas en coordenadas 3D, ya que solo hay que escalar las proyecciones a los ejes X e Y de la escena, dejando la Z a cero. Sin embargo en cualquier otra geometría resulta complicado saber con exactitud donde se encuentra cada punto 3D. Para solucionar esto, se ha implementado un *raycaster*.

Un *raycaster* consiste en un test de intersección rayo-superficie útil para resolver problemas en el ámbito de los gráficos por ordenador o en geometría computacional.

En este caso permite lanzar rayos desde la posición de cámara en una dirección obtenida a partir de la proyección hacia la escena, donde colisionará con los objetos representados en la imagen de referencia y de donde se extraerán los puntos de colisión y los vectores normales de esas superficies.

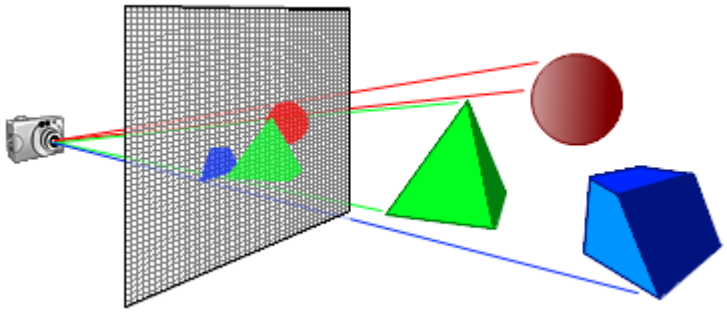


Ilustración 28 Raycaster

Como parámetro de entrada al *raycaster*, se introduce la posición 2D de la proyección, los parámetros de cámara (matriz de proyección y matriz de cámara), un nodo de la escena, distancia mínima y máxima y un parámetro de recursividad.

Si el parámetro de recursividad está activo, el *raycaster* se ejecutará para todos los hijos de ese nodo.

Cuando se ejecute para un nodo de tipo “mesh” (un objeto 3D), el *raycaster* comprobará si hay colisión entre el rayo y cada una de las caras de ese nodo, para ello lo primero que hará es calcular si el vector normal, en la dirección del rayo, tiene el mismo o diferente sentido, lo que representaría si colisiona contra una cara interior o exterior del objeto respectivamente.

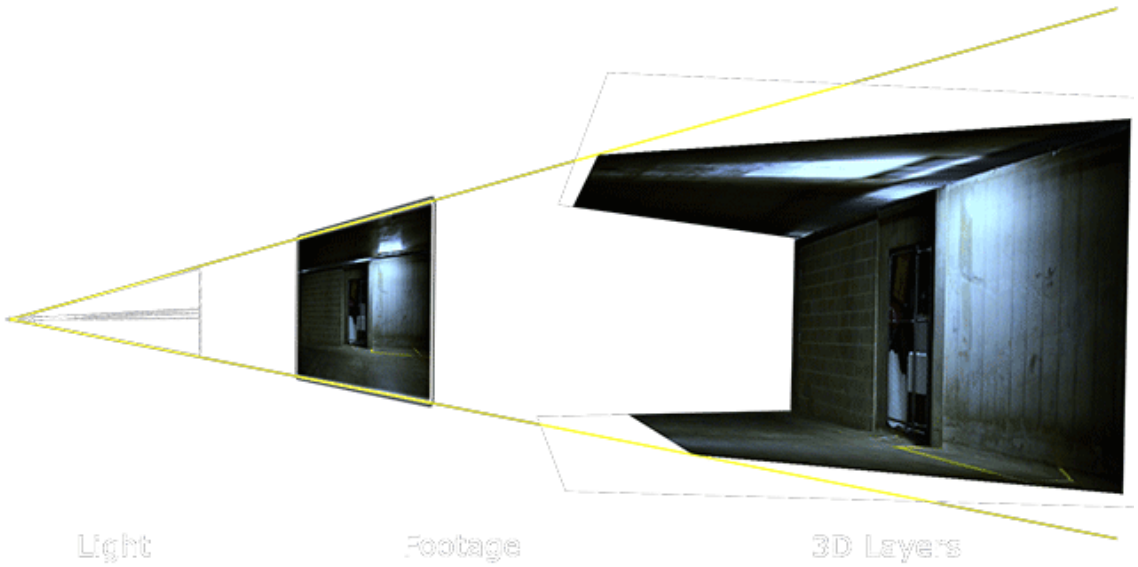


Ilustración 29 Proyección de escena en plano de imagen

Se filtrará para permitir colisiones solamente con las caras exteriores. Tras esto calcula el punto de colisión con el plano formado por los 3 vértices de esa cara y si el punto está dentro del triángulo formado por los tres vértices, almacenará una colisión con el punto 3D, el vector normal de la superficie y la distancia a la cámara.

Si la colisión está dentro de los límites marcados por la distancia máxima y mínima, entonces el parámetro de salida del *raycaster* devolverá un valor verdadero.

4.4 Cálculo de Vectores Normales

Para comprobar cómo se comporta el algoritmo he condiciones hostiles en las que las estimaciones en el cálculo de los vectores normales tendrán un error más o menos grande, se ha procedido desviando el vector de su posición original añadiendo ruido.

Puesto que en entornos reales se pueden dar casuísticas con características muy diferentes, se ha añadido el ruido de manera progresiva y observado los resultados para cada caso concreto.

La metodología seguida ha sido añadir ruido gaussiano al ángulo de inclinación del vector (en intervalos de 5 grados), con media centrada en el punto medio del intervalo (0, 7'5, 12'5, 17'5, etc.) y con una varianza igual un cuarto de la media, consiguiendo que el 96% de los vectores estén dentro de los 5 grados indicados por la media. Para el azimut, se ha utilizado una distribución uniforme, de forma cualquier dirección tenga la misma probabilidad que otra:

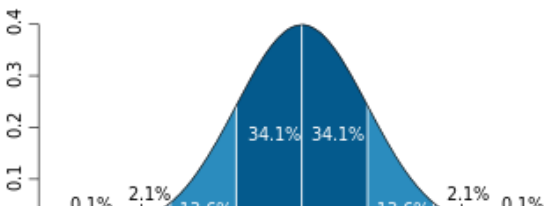


Ilustración 30 Distribución gaussiana

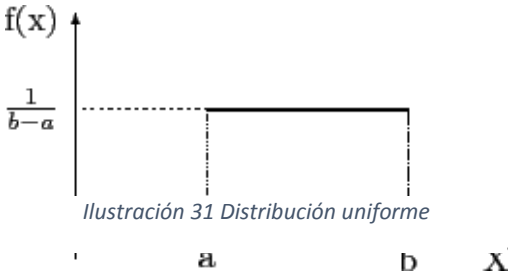


Ilustración 31 Distribución uniforme

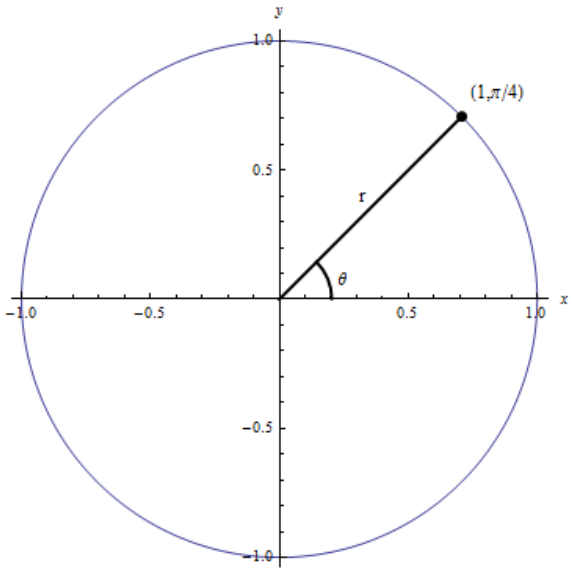


Ilustración 32 Distribución uniforme coordenadas polares

Realizando la composición de las dos distribuciones:

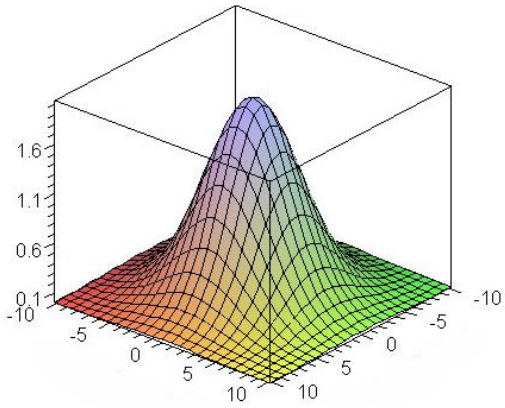


Ilustración 34 Curva gaussiana 3D

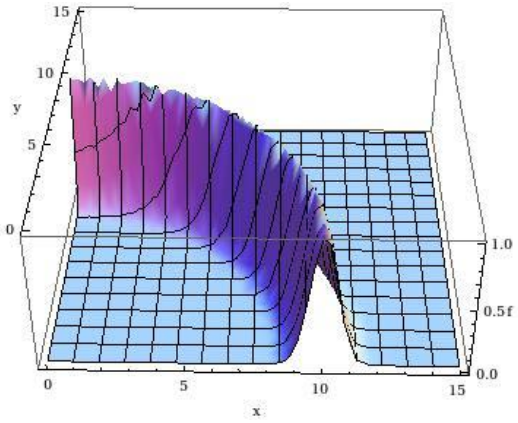
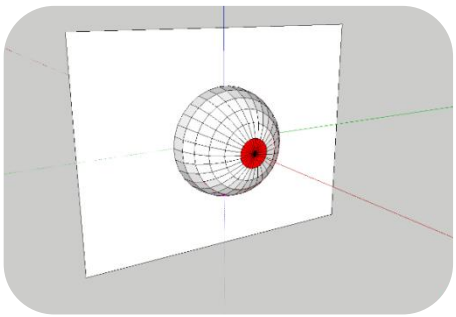
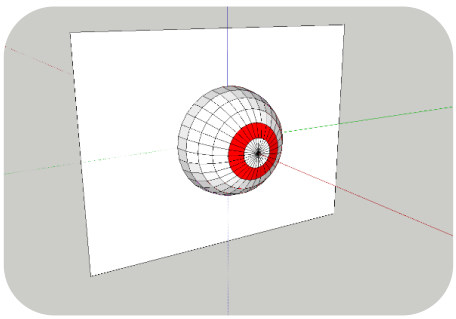


Ilustración 33 Convolución distribuciones gaussiana y uniforme

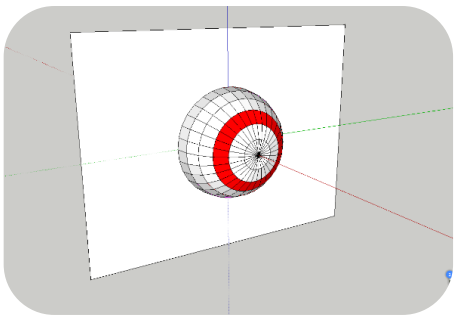
A continuación se muestra cómo se va variando la inclinación del vector, dentro de la semiesfera de todas las posibles posiciones, en intervalos regulares (ilustración aproximada):



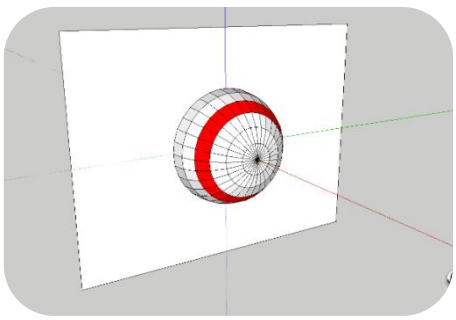
0-5º



5-10º



10-15º



15-20º

Ilustración 35 Variación de ángulo de inclinación

El resultado obtenido para los doscientos vectores en cada una de sus variaciones (por colores) se observa en Ilustración 36 e Ilustración 37:

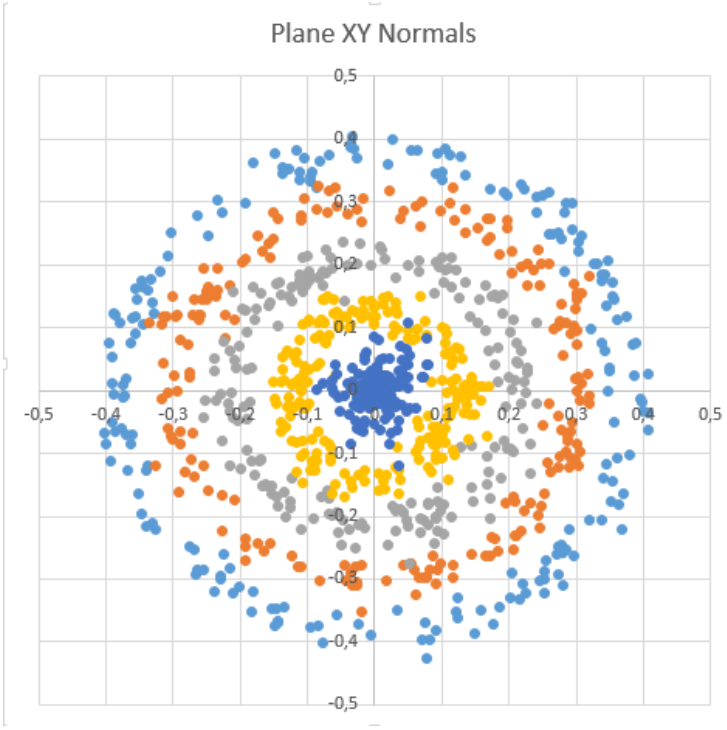


Ilustración 36 Resultado de variación de vectores normales (Vista superior)

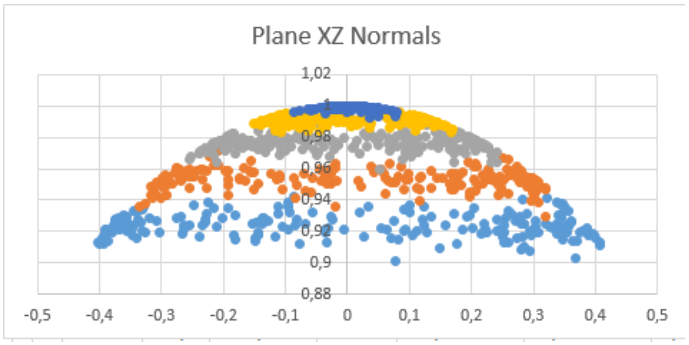


Ilustración 37 Resultado de variación de vectores normales (Vista lateral)

Para el caso del plano, la adición de ruido resulta muy sencilla ya que todos los vectores normales son iguales y están orientados en el eje Z ([0, 0, 1]). Por tanto se ha de variar la inclinación respecto a ese eje y el plano de rotación del azimut será siempre el plano X-Y.

El caso de la esfera resulta más complicado por el hecho de que cada punto tiene un vector normal distinto dependiendo de su posición en la esfera.

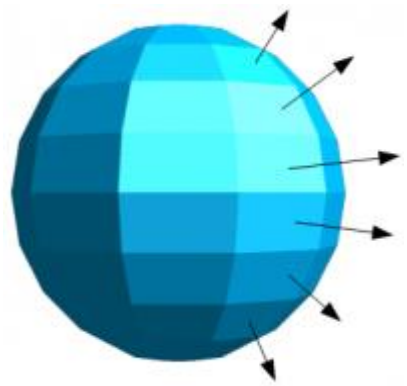


Ilustración 38 Vectores normales para representación de una esfera 3D

La solución en este caso, consiste en rotar el vector [0, 0, 1] hasta la posición de cada vector normal. Para hacer esto, el vector [0, 0, 1] por cada uno de los vectores, obteniendo un vector perpendicular a ambos cuyo módulo es igual al ángulo de separación entre ambos (en radianes). Sobre este vector se realiza la rotación y con el valor del ángulo se crea una matriz de rotación con la siguiente fórmula:

$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}.$$

Donde $\hat{u} = (u_x, u_y, u_z)$ es el vector y θ es el ángulo de rotación.

Tras calcular las doscientas matrices de rotación, se añade ruido al vector [0, 0, 1] y posteriormente se rota hasta la posición del vector con ruido.

Como punto de partida, la idea era comprobar como de importante es que los vectores normales estén bien calculados. Por ello en un primer momento, se plantearon las pruebas con una variación de los vectores totalmente aleatoria, es decir, cada vector podía apuntar a cualquier dirección. Sin embargo, los resultados obtenidos no arrojaban resultados fácilmente ponderables. Había puntos en los que el vector se había desplazado poco y por tanto daba un error aceptable y otros casos totalmente opuestos donde el error era totalmente aleatorio. Por esa razón se decidió cambiar el planteamiento e ir aumentando la desviación de los vectores gradualmente.

4.5 Cálculo estadístico de error

Con todos los archivos de texto obtenidos de cada una de las variaciones del vector normal en ambos escenarios, se han elaborado unas tablas en las que se muestra la información de manera ordenada y coherente.

Tras importar los archivos de texto en tablas separadas, se crea una tabla general en la que aunar todos los resultados.

Para cada una de las variaciones, se ha comparado la proyección obtenida con el algoritmo respecto a las proyecciones reales. El valor obtenido corresponde a la distancia euclídea entre ambas proyecciones:

$$Error = \sqrt{(PROYWaPT.X - PROYreal.X)2 + (PROYWaPT.Y - PROYreal.Y)}$$

Este dato representa el error que se comete tras reproyectar los puntos con los parámetros de cámara de la imagen anterior y corregirlos con el algoritmo.

Con esta información se calcula la media, mediana, desviación estándar y valor máximo y valor mínimo para cada variación, de forma que se puedan observar las diferencias entre ellas.

También se ha implementado una tabla que ordena los puntos en función de su distancia euclídea en tres dimensiones al eje óptico, es decir, la distancia entre cada punto y la línea que parte de la cámara hacia el punto central de proyección. Esto es útil para el caso de la esfera, ya que su centro es un punto del eje óptico, de forma que se consigue saber cómo de desviado esta su vector normal para cada punto respecto a la dirección eje en función de la distancia.

Resultados Capítulo 5

A lo largo de este apartado se mostrarán los resultados obtenidos en los experimentos para determinar cómo se comportará el algoritmo ante diferentes casuísticas.

En este análisis se mostrará cómo responde "WaPT" ante una entrada de datos erróneos y cuan fiable y robusto es el método.

Cabe recordar en primer lugar el funcionamiento del mencionado algoritmo. El sistema está compuesto por dos partes principales; la primera de reconstrucción 3D y la segunda de *tracking*. La primera se realiza en pre-proceso. Es la encargada de obtener la nube de puntos 3D que representa al entorno y del mismo modo, la encargada de estimar los vectores normales para cada uno de dichos puntos 3D.

Para ello, en el proceso de reconstrucción se requiere como entrada al sistema una o varias imágenes de referencia desde las que reconstruir la nube de puntos 3D y sobre las que se estiman sus vectores.

En el proceso de *tracking* (*online*), sin embargo, se carga la reconstrucción con sus vectores normales junto con las imágenes de referencia. Los vectores normales son usados en el proceso para modificar con precisión la perspectiva de cada parche que luego serán usados a modo de *feature* para calcular la matriz de proyección. Es por ello que los vectores normales cumplen un papel de vital importancia en este algoritmo.

A partir de los resultados de los experimentos aquí presentados, se quiere demostrar la importancia del cálculo de estos vectores y cuan preciso debe ser ese cálculo para resultar efectivo. Con dicho objetivo, los experimentos presentados valoran los resultados del *tracking* teniendo en cuenta el vector normal correcto y variaciones controladas del mismo.

Con el fin de valorar correctamente la importancia del vector normal, el resultado del proceso de reconstrucción 3d debe ser estricto. Por ello se ha simulado bajo un escenario digital con imágenes sintéticas.

En el caso de análisis, se ha utilizado una única imagen de referencia de entrada ya que se conoce la posición exacta de todos los puntos 3D y no es necesario calcularla. En cuanto a los vectores normales, se ha ido variando de forma controlada respecto de su posición correcta.

Para variar esos vectores, como se explica en el apartado 4.4, se ha añadido ruido gaussiano de forma gradual a los vectores y se han creado varios archivos de reconstrucción con cada una de las variaciones.

Por otro lado, se ha introducido un archivo de video con dos *frames* en los que se ha variado la matriz de proyección de la cámara, consiguiendo una nueva perspectiva del entorno.

Para valorar la fiabilidad del método se ha observado como varía el punto de proyección calculado por el algoritmo tras deformar el parche de referencia y compararlo con el de

la imagen de entrada respecto a la proyección verdadera calculada en el modelo sintético.

Como se explica en 4.1 y 4.2, se han creado 2 escenarios diferentes, el primero con un plano y el segundo con una esfera. Para los 2 casos se ha colocado una cámara en la posición (0, 0, 40) apuntando hacia el objeto y se ha *renderizado* una imagen de referencia de ambos. De esta imagen se han extraído 200 puntos característicos de los que se ha calculado su posición 3D y sus vectores normales, creando el archivo de reconstrucción.

Los resultados se comentan para cada uno de los escenarios ya que poseen características diferentes.

5.1 Plano

Como se ha mencionado con anterioridad, los experimentos se han realizado bajo dos tipos de geometrías. En el apartado que nos ocupa, se detallan los resultados de dichos experimentos para el caso de la geometría de un plano acotado. Los resultados obtenidos se aprecian en la *Tabla 2*.

En la *Ilustración 1* se muestra el error que comete el algoritmo en el cálculo de las proyecciones de los puntos 3D respecto al verdadero valor que debería dar (de aquí en adelante se denominará a este error como Error de reproyección). Se ha realizado una selección de 50 puntos. En la tabla, se muestran por filas la media, máximo, mínimo, media y desviación estándar. Y por columnas, el error para cada una de las variaciones del vector normal (con variaciones incrementales) desde 0 hasta 40 grados respecto a la posición en 0, como se describe en 4.4. La última columna muestra el error que presenta el método sin haber deformado el parche, o lo que es lo mismo, como harían otros métodos clásicos.

Como se ha dicho anteriormente, de los 200 puntos se han escogido 50 uniformemente distribuidos.

Como se observa, el error se desvía de un valor "aceptable" (cercano a cero) en mayor cantidad de puntos conforme la normal se desvía más de cero.

Para el caso en el que no se deforma el parche (última columna) el error no sigue ningún patrón.

A continuación se muestra un resumen estadístico de los resultados y una representación gráfica de los mismos:

Pts.	N0	N0-5	N5-10	N10-15	N15-20	N20-25	N25-30	N30-35	N35-40	NNW
Avr:	0,9519	0,79	0,5198	0,4835	1,0798	0,9979	3,7909	11,899	11,851	20,08
Max	11,2	11,8	1,2	1,0	23,6	14,7	56,2	62,1	82,9	63,3
Min	0,2	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,3
Median	0,7	0,5	0,5	0,5	0,5	0,5	0,7	0,9	1,2	9,4
SDEVAT	1,5728	1,6678	0,239	0,2255	3,3164	2,1338	10,461	17,995	19,282	20,55

Tabla 2 Resumen estadístico Plano

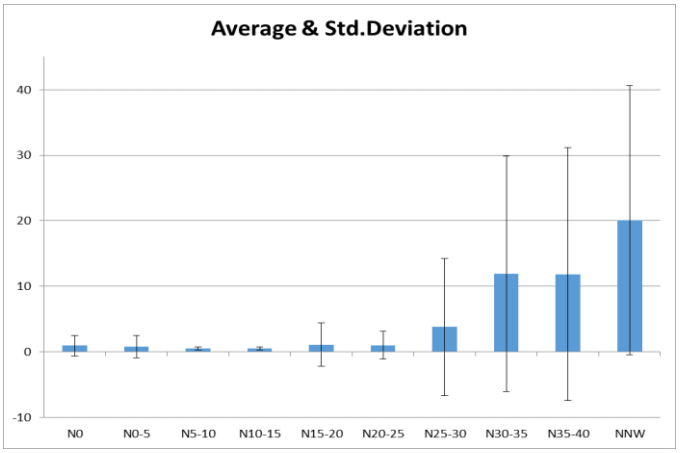


Ilustración 39 Gráfico de resultados Plano

5.2 Esfera

Para este caso se ha procedido como con el Plano, aunque las rotaciones y traslaciones han sido más leves, los ángulos de rotación en este caso han sido de 0º y 5º respecto del centro de la esfera. El hecho de aplicar una rotación más leve no infiere en el resultado ya que en una aplicación real se supone no se van a dar una variaciones tan bruscas entre *frames* de un video.

Los resultados obtenidos son los siguientes:

En la *Tabla 3*, de nuevo, se muestran los errores de reproyección tras corregirlos con "WaPT" respecto al resultado correcto, como en el caso del plano, se muestran por filas los valores estadísticos, como la media, máximo, mínimo, mediana y desviación estándar. Por columnas se muestran también las variaciones de manera incremental desde 0º a 40º.

Pts.	N0	N0-5	N5-10	N10-15	N15-20	N20-25	N25-30	N30-35	N35-40	NNW
Avr:	4,2	4,1	3,9	3,9	3,3	3,4	3,6	5,1	6,2	3,0
Max	44,9	44,9	44,9	37,7	35,9	37,5	43,5	43,0	62,6	30,8
Min	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
Median	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,4
SDEVAT	10,031	9,9056	9,8188	8,5459	8,0787	8,3924	8,8726	11,479	14,407	6,3197

Tabla 3 Resumen estadístico Esfera

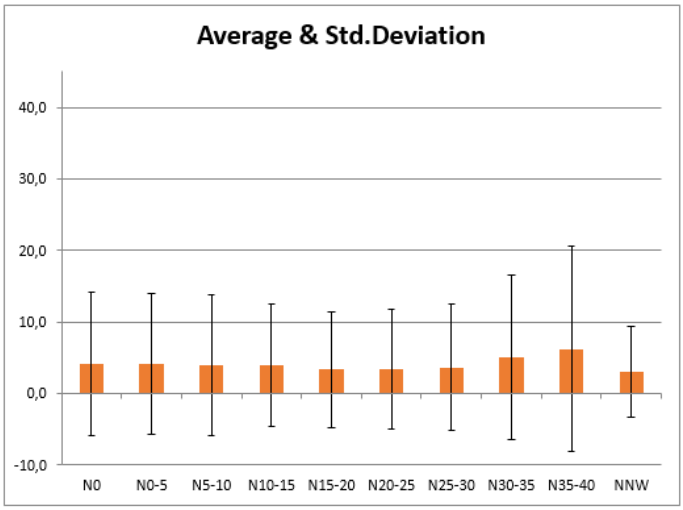


Ilustración 40 Gráfico de resultados Esfera

El comportamiento en este caso es más errático. Sí que se observa un incremento conforme la deformación de la normal se hace más grande, pero hay cierto comportamiento extraño no exento de explicación.

Si se ordenan los resultados de forma que se posicionan los puntos por la distancia de estos al eje óptico (punto central de la imagen) de menor a mayor, se obtiene la siguiente resultados (Tabla 4).

Pts.	N0	N0-5	N5-10	N10-15	N15-20	N20-25	N25-30	N30-35	N35-40	NNW
Avr:	4,8	4,7	4,5	5,0	3,8	3,9	4,7	4,8	8,4	3,2
Max	44,9	44,9	44,9	37,7	35,9	37,5	43,5	43,0	62,6	30,8
Min	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
Median	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,4
SDEVAT	10,877	10,751	10,672	9,611	7,6586	7,8443	9,9974	10,766	14,772	6,7235

Tabla 4 Resumen estadístico Esfera ordenado

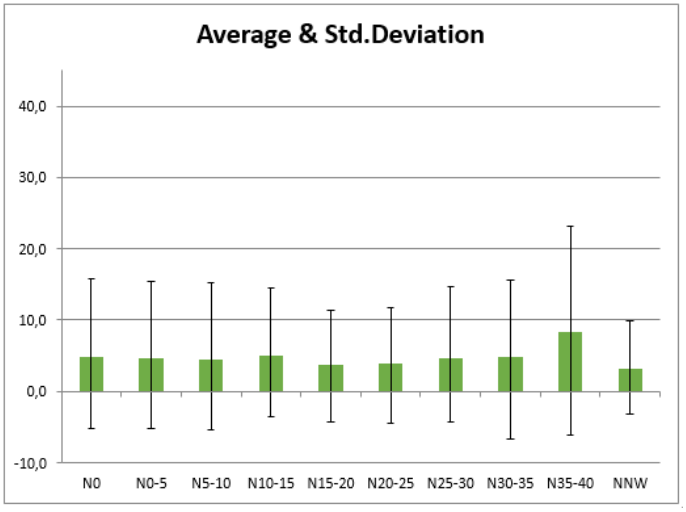


Ilustración 41 Gráfico de resultados Esfera ordenado

Los puntos del principio de la tabla son aquellos que están más cercanos al eje óptico, es decir, los puntos centrales de la esfera. Mientras que los últimos son los puntos del borde. La razón respecto a por que los puntos del borde tienen más error se explica por un error de paralaje al deformar el parche.

Una esfera en proyección siempre va a estar representada por un disco circular. Por tanto si se toma un parche del borde de la imagen de referencia y se deforma, se deja de observar un borde correspondiente a un círculo para pasar a observar un borde de una elipse (*Ilustración 142*). Al realizar la correlación cruzada de este parche deformado con la imagen, para ver a qué punto pertenece, este parche elíptico no casa de manera tan correcta como un parche con borde circular, que sería el resultado obtenido en la última columna para un parche que aunque si deformar, mantiene ese borde y por tanto ofrece una correlación cruzada más definida.

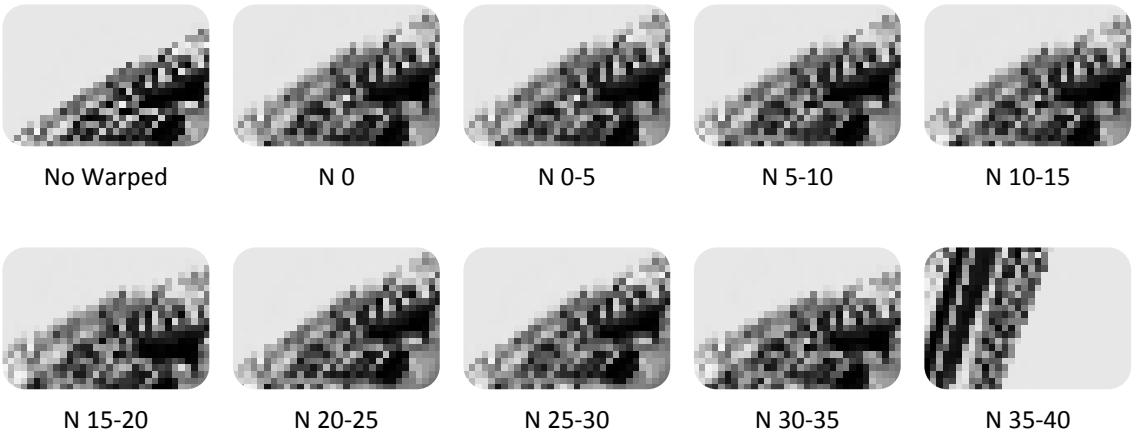


Ilustración 42 Parches deformados

Si solo se tienen en cuenta los puntos que no pertenecen al borde, los resultados estadísticos mejoran:

Puesto que los giros entre *frames* son leves, los resultados para los parches sin deformar (última columna) no son tan malos como en el caso del plano, sin embargo, los resultados se podrían asemejar a unos parches deformados con un error de 10-20 grados.

Parte Tercera

CONCLUSIÓN

Conclusiones

A la vista de los resultados, se puede afirmar que el algoritmo consigue reducir el error de cálculo de las proyecciones de los puntos cuando los vectores normales están correctamente calculados (dentro de ciertos límites), además, necesita menor cantidad de puntos que otros algoritmos, lo que deriva en un mejor cálculo de la posición de la cámara que es el objetivo último en este tipo de aplicaciones.

Sí es cierto que aunque necesita menor cantidad de puntos, hay que almacenar los vectores normales, lo que implica el doble de uso de memoria. Sin embargo esa cantidad de puntos está por debajo de la mitad respecto a una aplicación convencional, por lo que se mejora en este aspecto.

Por otro lado, hay que tener en cuenta que los escenarios analizados son escenarios de prueba y su complejidad no es comparable con un escenario real.

Por otro lado, comparando el caso del plano y de la esfera, se puede afirmar que el algoritmo funciona mejor en puntos que se encuentran en superficies planas y amplias, de forma que el parche a su alrededor se encuentre totalmente dentro la superficie. Esto también es así para el resto de los algoritmos. Sin embargo en este caso los resultados mejoran sobremanera.

Como se ha observado, siempre habrá problemas en puntos que pertenezcan a bordes reales dentro de la escena debido a que los elementos detrás de esos bordes variarán mucho más que el borde cuanto más lejos se encuentren del mismo y cuanto más pronunciado sea el movimiento de la cámara.

Por ello, los escenarios elegidos permiten sacar en claro muchas conclusiones, sin embargo al no tratarse de entornos reales también tienen carencias. Por ejemplo la ausencia de fondo texturizado. Esto provoca que, como se observa en el caso de la esfera, un algoritmo clásico dé un mejor resultado que “WaPT” debido a que, como se ha comentado en el apartado 5.2, al deformar la imagen se deforma la forma del borde también, lo que provoca que un parche de un algoritmo clásico, el cual no deforma la perspectiva, de un resultado de correlación cruzada más definido que el de “WaPT” simplemente por el hecho de que el fondo de la imagen detrás de la esfera, es un color plano. Lo que no queda claro es si con un fondo no plano, los resultados mejorarían para “WaPT”.

Trabajo Futuro

Como trabajo futuro, se plantean varias cuestiones. Por un lado el uso de un entorno digital controlado, sirve para cuantificar los límites del algoritmo en cada uno de sus parámetros. Sin embargo, la aplicación real del mismo está pensada para entornos reales. Es por ello que en primer lugar, habría que analizar su comportamiento con un entorno controlado, pero real. En el que se conozca la posición de los puntos de una forma lo más precisa posible mediante herramientas de precisión.

Por otro lado, en este TFG no se ha utilizado la parte de reconstrucción para el cálculo de la nube de puntos y sus vectores normales, si no que se conocen de antemano. Así pues, habría que comprobar como de robusto es este método de cálculo de la nube de puntos con sus vectores normales para ese escenario real comentado en el párrafo anterior y compararlo con los resultados obtenidos con las herramientas de precisión. De forma que se pueda comparar la precisión de la fase de reconstrucción 3D.

En cuanto a la fase de reconstrucción de "WaPT" también, como se comenta en el capítulo 3, a cada punto de la nube de puntos se le asigna un *keyFrame* en el cual se ha visto al punto por primera vez. Sería óptimo cambiar esa condición para hacer que a cada punto se le asigne el *keyFrame* en el que tenga su vector normal orientado más paralelamente al eje óptico del *keyFrame*. Ya que, como se ha comprobado en el caso de la esfera, apartado 5.2, se consigue disminuir el error para los puntos con esta característica. Esto es debido a que, lógicamente, si su vector normal está orientado de forma paralela al eje óptico, la cámara capta la superficie en la que se encuentra el punto de forma totalmente perpendicular, es decir, sin deformaciones de perspectiva. Esto provoca que al deformar el parche en la fase de *tracking*, se evite una acumulación de error debido a problemas de paralaje.

Por último, si todas las pruebas descritas anteriormente arrojan unos resultados positivos, faltaría aplicar el algoritmo en condiciones de "usuario", es decir, con un dispositivo móvil en un entorno real. Esto implica, dependiendo del dispositivo, que la cámara puede tener diferentes calidades de captura, lo que en fase de reconstrucción introduciría un ruido desconocido, de forma que la nube de puntos tendría un ruido variable, dependiendo de esa calidad de la cámara. En estas condiciones, tiene especial importancia la parte de *Bundle Adjustment* y los filtros para eliminar los resultados erróneos.

Referencias

[1] Milgram, P., & Kishino, F. (1994). A taxonomy of mixed reality visual displays.*IEICE TRANSACTIONS on Information and Systems*, *77*(12), 1321-1329.

[2] Azuma, R. T. (1997). A survey of augmented reality. *Presence*, *6*(4), 355-385.

[3] Davidson, A.J., Reid, I.D., Molton, N.D. & Stasse, O. (2007). Monoslam: Real-time single camera slam. *In Pattern Analysis and Machine Intelligence*, IEEE Transactions on, volume 29, pages 1052-1067. IEEE.

[4] Barrena, N., Sánchez, J.R. & García-Alonso, A. (2015). Surface normal estimation for improved template matching in visual tracking.

[5] Koenderink, J.J. & Van Doorn, A.J. (1991). Affine structure from motion. JOSA A, 8(2):377-385.

[6] Harris, C. & Stevens, M. (1988). A combined corner and edge detector. In Alvey vision conference, volume 15, page 50.

[7] Rosten, E. & Drummond, T. (2006). Machine learning for high-speed corner detection. En Computer vision-ECCV 2006, pages 430-443.

[8] Alahi, A., Ortiz, R. & Vandergheynst, P. (2012). Freak: Fast retina keypoint. En Computer vision and pattern recognition (CVPR), 2012 IEEE conference on, pages 510, 517.

[9] Triggs, B., McLauchlan, P.F., Hartley, R.I. & Fitzgibbon, A.W. (2000). Bundle Adjustment – A modern synthesis. En Vision algorithms: theory and practice, pages 298-372.

[10] Hartley, R. & Zisserman, A. (2000). Multiple View Geometry in computer vision. Second Edition.

[11] Davidson, A. J. (2003). Real-time simultaneous localization and mapping with a single camera. In Computer Vision. Ninth IEEE international Conference on, pages 1403-1410.

[12] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. 60:91-110.

[13] Bay, H., Tuytelaars, T. & Van Gool, L. (2006). Surf: Speeded up robust features. In Computer Vision ECCV 2006, pages 404-417. Springer.

[14] Furukawa, Y. & Ponce, J. (2010). Accurate, dense and robust multiview stereopsis. 32(8):1362-1376.