Tesis de Máster

Reconstrucción densa de modelos tridimensionales utilizando Visión Artificial

Unai Mujika Torrontegi

Donostia, Septiembre de 2010

Universidad del País Vasco / Euskal Herriko Unibertsitatea
Departamento de Ciencia de la Computación
e Inteligencia Artificial

Director: Basilio Sierra Araujo
Jon Azpiazu (Teknalia)

www.ccia-kzaa.ehu.es

Estudio del estado del arte de visión estéreo

Índice de contenido

Introducción:

La visión es uno de los 5 sentidos del ser humano, y podría decirse que es el más importante de los mismos. Este sentido nos permite obtener grandes cantidades de información sobre nuestro entorno, datos que son esenciales para nuestra supervivencia. Gracias a ella somos capaces de identificar en un momento objetos, animales y semejantes situados dentro de nuestro campo visual, estimar su posición para que a continuación nuestro cerebro pueda valorar su utilidad e importancia. Pero este también es uno de los sentidos más complicados de reproducir artificialmente, debido a que la psicología cognitiva no ha sido capaz de entenderlo completamente [15].

Conocemos como visión artificial a la capacidad de un sistema artificial, robótico o no, de percibir el espacio que les rodea mediante la visión. A día de hoy, los sistemas de visión artificial no son capaces de percibir su entorno con la misma precisión y completitud que sus análogos humanos o animales.

La miniaturización de los equipos permite desde hace tiempo situar cámaras y sistemas de procesado en plataformas móviles y en robots, y su abaratamiento ha hecho que cada vez sean más comunes en entornos como líneas de producción. En este tipo de sistemas se han utilizado para tareas como el mapeado del entorno, o el reconocimiento de patrones [16].

En el mapeado del entorno, encontramos técnicas como el SLAM que son capaces de detectar la posición en el espacio físico de unos pocos puntos fácilmente reconocibles. El problema de este tipo de técnicas es que tan solo perciben una pequeña parte de la escena, y es posible que partes importantes de la misma no sean analizadas.

En el reconocimiento de patrones y objetos, hay que destacar el problema conocido como "*Head Pose Estimation*", que busca estimar la pose, posición + orientación, de un objeto conocido y que en el ámbito de este problema suponemos no deformable.
Se han realizado múltiples aproximaciones a este problema [18], sin que a día de hoy sea posible estimar con precisión la pose de la cabeza humana, pero citamos aquí este problema debido a la gran cantidad de algoritmos que se han generado, y a la posibilidad de trasladar estos algoritmos a otros problemas de estimación de pose y reconocimiento de objeto.

Dentro de los campos pertenecientes a la visión artificial este proyecto se centra en la reconstrucción automática del entorno, con la idea de poder utilizarlo en sistemas de robótica, de tal forma que estos sean capaces de reconocer su entorno mediante la visión y generar mapas de su entorno. Para ello queremos que el sistema sea capaz de realizar la reconstrucción tan rápido como sea posible, y que la reconstrucción sea lo más completa posible (ver apartado sobre reconstrucción densa para más detalles). Además, para poder generar mapas, vamos a querer que el sistema sea capaz de relacionar la escena que ve a cada momento con la escena que ya ha visto, para de esta forma poder generar reconstrucciones que abarquen áreas aun mayores.

Objetivos del proyecto:

Este proyecto se plantea como objetivo el realizar una reconstrucción densa del entorno del sistema de visión artificial mediante el uso de múltiples imágenes. Además, se desea que el sistema sea capaz de realizar la reconstrucción en tan poco tiempo como sea posible, para facilitar su implantación en sistemas robóticos.
Inicialmente se realizará un estado del arte de las técnicas de reconstrucción mediante visión existentes, y posteriormente se implementará una de ellas.
Los hitos del proyecto son:

1. Creación de un algoritmo de calibración para el sistema de cámaras estéreo.
2. Extracción de características discretas de las imágenes individuales.
3. Emparejamiento automático de puntos de interés.
4. Detección y filtrado de emparejamientos espúreos.
5. Reconstrucción discreta del entorno a partir de los puntos de interés.
6. Reconstrucción densa del entorno a partir del emparejamiento denso de puntos entre imágenes.
7. Cálculo automático de desplazamiento y rotación del sistema de cámaras entre frames.
8. Fusión de reconstrucciones obtenidas por el sistema de cámaras en los distintos frames.

Trabajo Realizado:

En este proyecto se ha buscado generar un sistema estéreo capaz de generar reconstrucciones densas de escena en tiempo real, o casi tiempo real. Los términos utilizados en esta sección se explican más adelante en este documento.

Para poder hacerlo se ha utilizado la librería OpenCV, con la que se han creado los siguientes programas:

-Calibración del sistema estéreo: Programa que calcula las matrices de cámara de las webcam utilizadas (Logitech C905) y la rotación/traslación entre ellas. Devuelve la estimación de error de reproyección de la solución obtenida, y las matrices M, R, T, E y F del sistema, almacenándolas en ficheros para su uso en el resto de los programas.

Para realizar la calibración se obtienen 20 pares de imágenes estéreo en las que se localizan todas las esquinas de las celdas negras de un tablero de ajedrez a un nivel sub-píxel. A continuación se utiliza el algoritmo de calibración estéreo incluido en OpenCV para obtener las matrices del sistema y el error de reproyección.

-Evaluación de algoritmos de correspondencia: Programa que permite utilizar los distintos algoritmos de correspondencia presentes en OpenCV (BM, SGBM y GC) y comparar su rendimiento en base a % de completitud del mapa de disparidad y tiempo de cálculo.

Este programa carga de ficheros xml los datos de las matrices del sistema calculados en el programa de calibración. Tras esto, calcula los mapas de rectificación mediante una implementación del algoritmo de Bouguet incluida en OpenCV y calcula los mapas de rectificación para cada una de las imágenes.

Para cada frame capturado por el sistema estéreo, se rectifican las imágenes, y se buscan correspondencias. Pasado un segundo, se mide el número de frames procesados, y en el caso de que el algoritmo no haya terminado, se espera a que termine (caso de GC) y se mide su tiempo.

Para evaluar el % de completitud, se lee el mapa de disparidad generado, y se calcula el total de puntos para los que no se ha obtenido un valor válido. Hay que tener en cuenta que se ha configurado los algoritmos de correspondencia para que devuelvan valores sin sentido cuando no sean capaces de obtener una correspondencia para un píxel.

-Programa de reconstrucción 3D: Programa que toma imágenes del sistema de cámaras de forma simultanea generando un par estéreo y mediante triangulación realiza una reconstrucción densa de la escena.

Al igual que el anterior, carga las matrices del sistema desde ficheros, y calcula los mapas de rectificación mediante el algoritmo de Bouguet.

Para cada frame capturado por el sistema estéreo, se rectifican las imágenes y se buscan correspondencias mediante SGBM. Tras esto se reconstruyen los puntos de la escena mediante triangulación.

-Programa de reconstrucción 3D multi-view: Programa, inacabado, que genera reconstrucciones de la escena a cada frame, y que une las reconstrucciones de todos los frames para generar una reconstrucción mayor.

Al igual que los anteriores carga las matrices del sistema desde ficheros, y calcula los mapas de rectificación mediante el algoritmo de Bouguet.

Tras esto rectifica las imágenes y busca mediante el algoritmo SIFT puntos de interés en la imagen izquierda, y se almacenan sus descriptores en un kdd tree.

De la misma forma que en el programa de reconstrucción se calcula el mapa de disparidades y la reconstrucción de la escena, pero tras ello, si hemos procesado más frames, se intenta relacionar el frame actual con el frame más similar. Para ello se comparan los descriptores SIFT de la imagen izquierda del frame actual con los de los últimos 15 frames. Para ello buscamos correspondencias entre los descriptores de uno y otro frame, y desechamos aquellos que devuelvan una disparidad superior a 100 píxeles para evitar posibles falsos positivos.

A la hora de decidir con que frame vamos a emparejar el frame actual elegimos aquel con mayor número de correspondencias, y con menor sumatorio de disparidades. Tras esto utilizamos la librería de Bundle Adjustment [33] en su versión 1.5 para buscar la rotación y traslación que se ha producido desde la pose de la cámara izquierda del frame antiguo a la pose de la cámara izquierda del frame actual.

En la actualidad se encuentra pendiente el realizar la fusión de las reconstrucciones generadas en cada frame.

Cámara:

La cámara es el elemento básico que nos va a permitir percibir el entorno mediante la visión artificial. Consta de lente y sensor.

Óptica:

En una cámara el sensor es el elemento capaz de convertir la luz que le llega en impulsos eléctricos, y por tanto el que permite a la cámara visualizar el entorno, pero para que la luz de la escena pueda llegar al sensor, es necesario recogerla y dirigirla hacia el sensor, y de esto se encarga la lente.

En el modelo más sencillo de cámara, tomamos la lente como un único punto que es atravesado por los rayos de luz de la escena, sin provocar ninguna alteración de dirección en ellos. Llamamos a este modelo *Pinhole-model*.

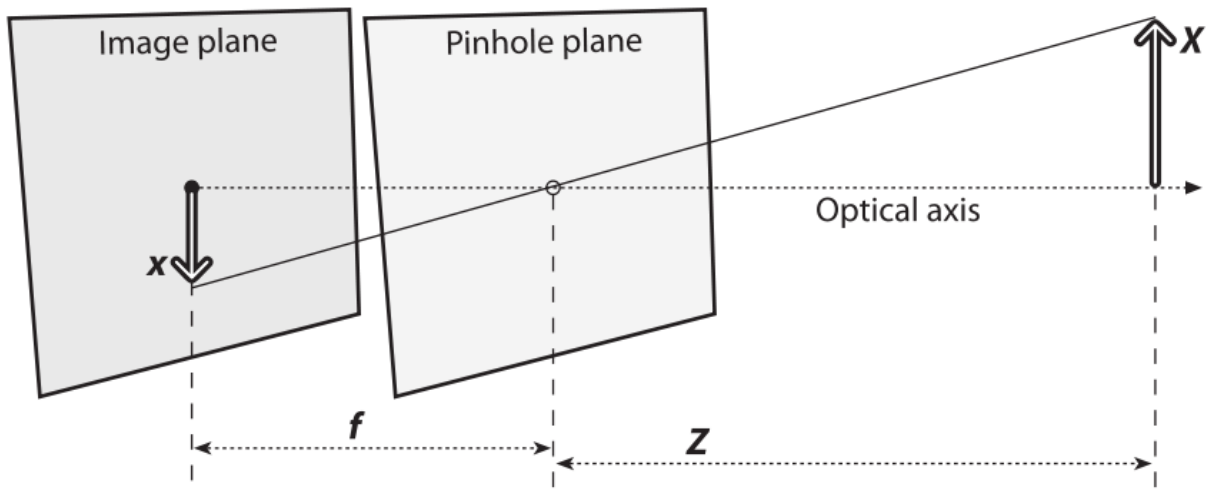

Ilustración 1: Pinhole model

Para la mayoría de los cálculos basta con considerar a la lente como un agujero puntual a través del que pasa la luz a través de un centro de proyección común, pero si vamos a tratar aspectos como la distancia focal o las aberraciones de la lente, es necesario utilizar un modelo más complejo, y aquí es donde entra en juego la óptica.

Digamos que vamos a visualizar un objeto que se encuentra a una distancia x_1 de la lente, y que la imagen proyectada por los rayos que salen de la lente se forma a una distancia x_2 tras la lente. Llamamos distancia focal a la relación que existe entre x_1 y x_2 :

$$\frac{1}{f}=\frac{1}{x_1}+\frac{1}{x_2}$$

Así mismo, una lente no tiene por qué producir una proyección uniforme sobre el sensor. Es posible, y muy probable, que debido a pequeños defectos de producción de la lente, esta produzca irregularidades en la proyección de la imagen. Llamamos a estos defectos de la lente aberraciones o distorsiones. Si bien existen múltiples tipos de aberraciones de la lente, nosotros vamos a centrarnos en la radial, que es una de las más frecuentes. Esta distorsión será tratada en el apartado de calibración.

Sensor Óptico:

La luz que pasa por la lente llega a la superficie del sensor óptico, que es el encargado de convertir esta luz en impulsos eléctricos.

En la actualidad hay principalmente dos tipos de sensores, CCD y CMOS, si bien CMOS es el más utilizado en las cámaras fotográficas y web de hoy en día, y el tipo de sensor que es utilizado en las cámaras usadas en este estudio.

Para convertir la luz que llega al sensor en pulsos eléctricos se establece un tiempo de exposición durante el cual el sensor recoge la luz y a continuación entrega la información a un amplificador. A este tiempo se le llama velocidad de obturación. En las cámaras de fotos analógicas la velocidad de obturación es el tiempo durante el cual el obturador está abierto y la película fotosensible está expuesta a la luz, y en los sistemas digitales sirve para marcar cada cuanto se muestrea la información (luz) que está llegando al sensor.

El sensor está dividido en pequeñas células fotosensibles, cada una de las cuales genera información que luego se traslada a un píxel. Cuanto mayor sea la cantidad de células en un área, mayor será la densidad de muestreo (sampling density) y mayor será resolución que podrá mostrar el sensor, y será menor el aliasing de la imagen. En contraste, mayor densidad significa que a cada célula llega menos luz, por lo que el sensor es más susceptible al ruido.

No todos los sensores tienen el mismo tamaño, y esto tiene una incidencia directa en la calidad de la imagen obtenida por el sensor. Si vamos a tener una determinada resolución, es preferible que las células fotosensibles sean de un tamaño lo suficientemente grande como para que les llegue suficiente luz.

Se conoce como ganancia del sensor a la operación analógica realizada por el amplificador para aumentar el nivel de la señal recibida. Esto puede ser útil a la hora de trabajar en condiciones de poca iluminación, ya que aumenta la intensidad de la luz percibida. El problema es que al ampliar la señal también se amplifica el ruido contenido en la misma. Hoy en día existen niveles de ganancia estandarizados (ISO 100, 200, 400...) para regular la ganancia que se va a aplicar a la imagen.

A la hora de obtener una imagen y decidir el nivel de ganancia que se le va a aplicar, es importante tener en cuenta el nivel de ruido presente en la misma. Para ello se genera una *noise level function*, que nos permite estimar cual será el nivel de ruido para cada canal de color en cada píxel de la imagen. De esta forma se pueden corregir las diferencias de sensibilidad entre diferentes partes de un sensor.

Sistemas de Coordenadas:

A la hora de representar los objetos del espacio observado (3D), los convertimos a un plano de visión (2D), y tenemos que definir cual es la correspondencia entre las coordenadas del espacio y las de la representación. Para facilitar esta conversión, se utiliza el sistema de coordenadas "homogéneo". Este sistema de coordenadas tiene la particularidad de que permite pasar fácilmente coordenadas de un número de dimensiones a otro. Para ello, almacena las coordenadas con una dimensión adicional, de tal forma que para un espacio de 3D, utilizaríamos 4 coordenadas. El valor de la coordenada adicional indica entre otras cosas, si el punto se encuentra en el infinito, valor 0, o es un punto cualquiera, valor distinto de 0. En este sistema, si dos coordenadas son proporcionales, se refieren al mismo punto.
Las matrices de transformación, utilizadas para pasar de unas dimensiones a otras, suelen incluir 3 tipos de transformaciones: **R**otación, **T**ranslación y e**S**calado. Podemos representarlas así:

$$Transformación = \begin{bmatrix} SR_{1,1} & SR_{1,2} & SR_{1,3} & T_1 \\ SR_{2,1} & SR_{2,2} & SR_{2,3} & T_2 \\ SR_{3,1} & SR_{3,2} & SR_{3,3} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Para convertir las coordenadas del mundo a coordenadas de pantalla, de espacio tridimensional a bidimensional, los sistemas de visión artificial utilizan la siguiente fórmula:

$$q = MQ \qquad \text{donde} \quad q = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad , \quad M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad \text{y} \quad Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Por tanto, q representa el espacio bidimensional en coordenadas homogéneas, y Q el espacio tridimensional. f_x es la distancia focal en el eje x, f_y la distancia focal en el eje y, (c_x, c_y) marcan el punto principal, el punto donde el eje de visión corta el plano de visión (normalmente suele estar en el centro de la imagen, o muy cerca).
Tanto la distancia focal como el centro están expresadas en píxeles. En el caso de querer obtener la distancia focal en unidades físicas, es necesario utilizar la siguiente fórmula:

$$f = \frac{W}{2}\left[\tan\frac{\theta}{2}\right]^{-1}$$

Donde W es la anchura física del sensor óptico y θ es el ángulo del campo de visión de la cámara.

Calibración:

Distorsión de la imagen:

El primer problema con el que nos encontramos al trabajar con sistemas de visión, es obtener una imagen no distorsionada. Debido a aberraciones en la lente, o defectos de construcción de la cámara, la imagen puede estar distorsionada.
Las dos distorsiones más comunes son la distorsión radial y la tangencial.

Distorsión radial:

Se llama así a las aberraciones producidas por defectos de construcción en la lente. Estos defectos suelen causar que cuanto más se aleje el punto de impacto del rayo de luz incidente del centro de la imagen, mayor será la desviación a la que será sometido. Esta clase de errores llevan a una visión de *ojo de pez*, en la que los píxeles situados en el centro de la imagen sufren pequeñas modificaciones, y los más alejados sufren aberraciones cada vez mayores. Por tanto podemos decir que el error acumulado por los píxeles de la imagen es directamente proporcional a la distancia a la que se encuentran del centro de la imagen, y que la distorsión de dos puntos situados a la misma distancia del centro, será similar. Por tanto utilizamos un modelo de circunferencias para representar estas aberraciones.
Una vez conocida la naturaleza de la aberración, podemos intentar corregirla. Para ello suelen utilizarse las siguientes ecuaciones:

$$x_{corregida} = x(1 + k_1 r^2 + k_2 r\,4 + k_3 r^6)$$
$$y_{corregida} = y(1 + k_1 r^2 + k_2 r\,4 + k_3 r^6)$$

En estas fórmulas, r representa la distancia desde el centro hasta la circunferencia de aberración donde se encuentra el punto a corregir, en otras palabras, el radio de dicha circunferencia. Los elementos k_1 , k_2 y k_3 son los coeficientes de corrección que vamos a utilizar. Estos coeficientes son valores que debemos hallar, ya que son los adecuados a la aberración concreta de nuestra cámara, y por tanto, son diferentes para cada cámara. El valor k_3 solo suele utilizarse en cámaras que presentan una gran distorsión radial, y en el resto suele dársele un valor de 0.
Tenemos funciones separadas para el eje x y el y debido a que los receptores de una cámara, que luego serán mapeados a píxeles, no son cuadrados, sino rectangulares, por lo que los parámetros de corrección para los dos ejes deberán ser diferentes.

Distorsión tangencial:

Se conoce con este nombre a la distorsión que se produce por la falta de paralelismo entre la lente de la cámara, y el sensor de la cámara, y genera imágenes trapezoidales. Se corrige utilizando las siguientes fórmulas:

$$x_{corregida} = x + [2\mathrm{p}_1 y + p_2(r^2 + 2\mathrm{x}^2)]$$
$$y_{corregida} = y + [2\mathrm{p}_1(r^2 + 2\mathrm{y}^2) + 2\mathrm{p}_2 x]$$

Los parámetros p_1 y p_2 representan los coeficientes con los que tenemos que alimentar a las funciones. Al igual que sucedía en la distorsión radial, los coeficientes de la función varían de una cámara a otra, por lo que deben de hallarse los coeficientes adecuados a la aberración de nuestra cámara.

Calibración en el proyecto:

A la hora de calibrar una cámara, tenemos que obtener 5 parámetros de distorsión, $k_{1-3}, p_{1,2}$, y 4 parámetros intrínsecos, f_x, f_y, c_x, c_y , para hallar la transformación que se debe llevar a cabo en la imagen.
Para hallar los 5 parámetros de distorsión, nos basta con una única imagen de calibración, en la que tengamos 3 puntos cuya posición sea conocida, pero es deseable utilizar más puntos para poder aumentar la robustez del cálculo.
El cálculo de los parámetros intrínsecos es más complejo. Primero, necesitamos saber la posición del objeto observado, típicamente un tablero de ajedrez, y para ello necesitamos 6 dimensiones, 3 para translación y otras tres para orientación (pitch, yaw y roll). En total, tenemos 10 (6+4) parámetros que necesitamos conocer. ¿Cuantas imágenes y cuantos puntos necesitamos para hallar para poder hallar todos los parámetros? Pues si tomamos N como el número de puntos y K como el número de imágenes: $(N-3)K \geqslant 2$ siempre y cuando $K > 1$.
Para obtener esta formula tenemos que tener en cuenta, que de una imagen obtenemos 2N datos (de cada punto o feat se obtiene un dato x y otro y), y que tenemos que resolver 4 parámetros intrínsecos y 6K extrínsecos (la posición del tablero en cada imagen). Por tanto necesitamos $2NK \geqslant 6K + 4$, y si simplificamos obtenemos la fórmula anterior.

A la hora de elegir un objeto para calibrar el sistema, se suele utilizar un tablero de ajedrez, en el que se utilizan como puntos para la calibración, los bordes en los que se tocan dos celdas negras. Es importante que el número de filas y columnas del tablero sea distinto, para que el sistema pueda interpretar la dimensión de variabilidad roll.

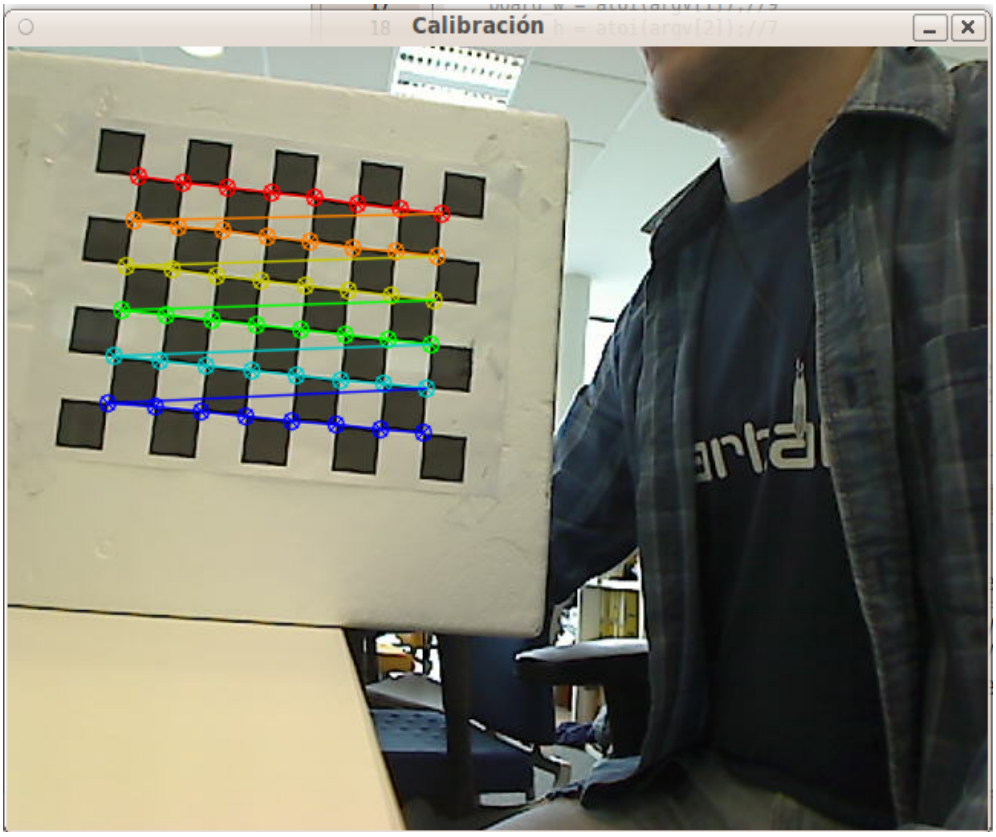

Ilustración 2: Puntos utilizados para calibración

Para que las imágenes sean útiles para la calibración, deben mostrar diferentes vistas del objeto de referencia, de lo contrario, es como si tuviésemos K=1.

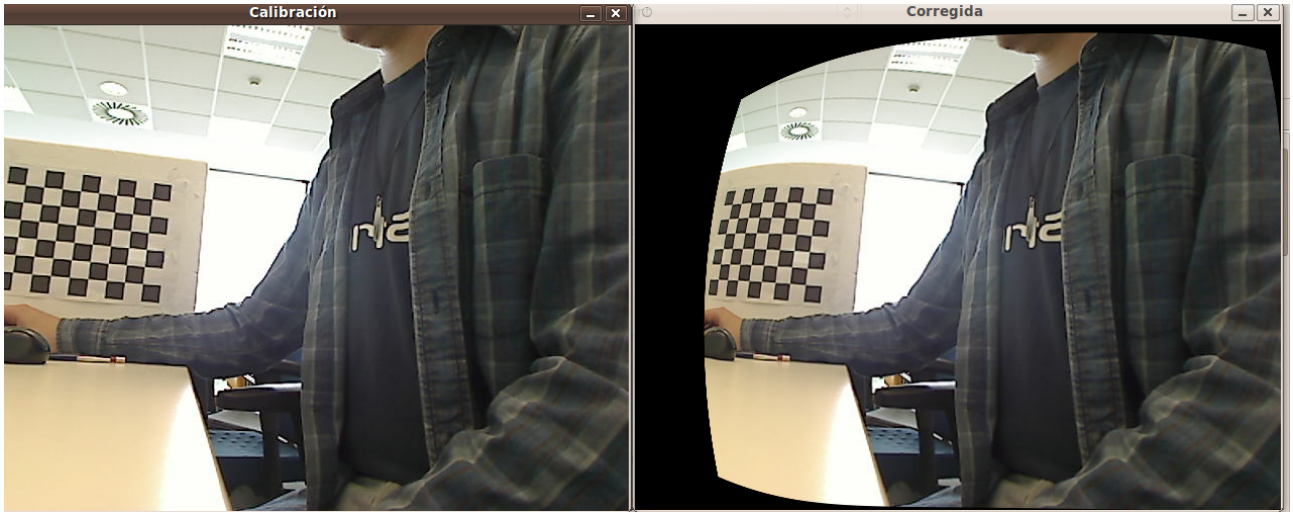

Ilustración 3: Resultado de la calibración sin utilizar vistas distintas del objeto de referencia

Si movemos el objeto de referencia y presentamos distintas vistas del mismo a la cámara, obtendremos una calibración en la que se corrijan los defectos de las aberraciones de la cámara.

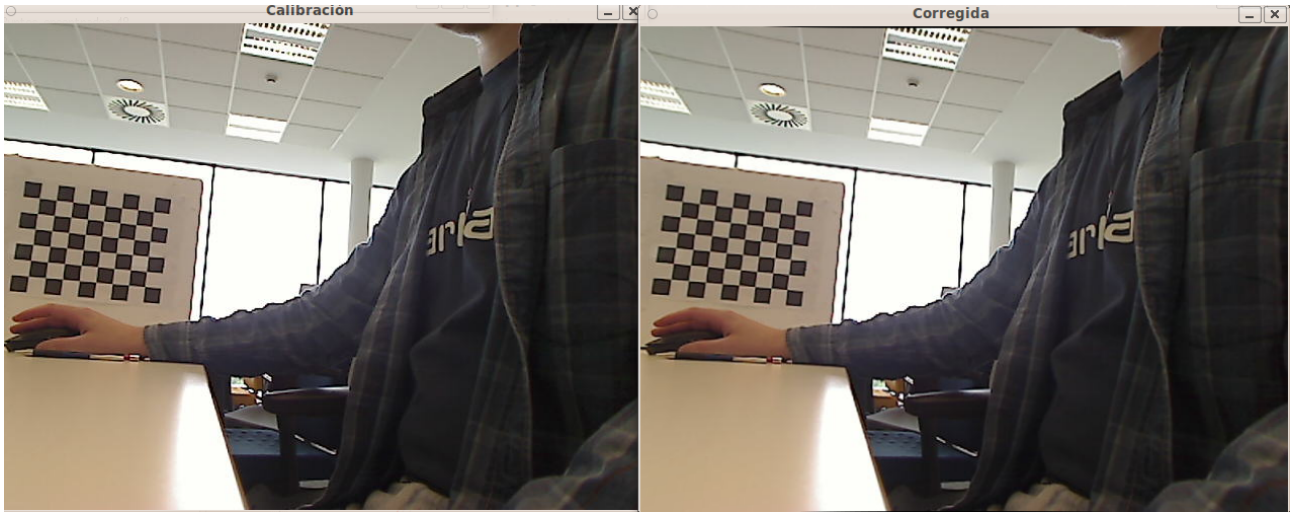

Ilustración 4: Vista original y corregida

En el proyecto se han utilizado 20 imágenes para obtener los parámetros de cámara. Estas imágenes son capturadas a razón de 1 por segundo, y a continuación se busca en cada una de ellas las esquinas de las celdas negras y se almacenan las coordenadas de los puntos detectados. Para la detección de los puntos hacemos uso de un algoritmo de detección de bordes Harris[19] implementado dentro de la plataforma de OpenCV.
A continuación se utiliza el algoritmo de calibración incluido dentro de OpenCV. Este algoritmo calcula una estimación inicial de los parámetros intrínsecos de la cámara y toma como 0 los valores de distorsión. A continuación se utiliza un algoritmo de Levenberg-Marquardt en el que se intenta minimizar una función de error en la que el error es la diferencia entre la posición de los puntos y la que resultaría de la proyección de estos mismos puntos utilizando los parámetros intrínsecos y extrínsecos calculados.
El error final de la función de calibración es utilizado en el proyecto como métrica de calidad de una calibración.

Para realizar la corrección mostrada en la ilustración nº4 el sistema ha calculado los siguiente parámetros:
Parámetros de distorsión:

$$K_1 = 0.06838191 \qquad K_2 = -0.04246903 \qquad K_3 = -0.55924994$$
$$P_1 = 1.41188775\text{e-}05 \qquad P_2 = 7.15511385\text{e-}04$$

Parámetros intrínsecos:

$$\text{M} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 545{,}66 & 0 & 306{,}26 \\ 0 & 544{,}63 & 239{,}55 \\ 0 & 0 & 1 \end{bmatrix}$$

Es interesante comprobar que los valores c_x, c_y se encuentran cerca del centro de la imagen (640x480), y que las distancias al plano focal son casi iguales en el eje ***x*** y en el eje ***y***, y que por ello los valores de P_1, P_2 son muy pequeños.

Estimación de pose en 3D:

Uno de los problemas clásicos de visión por computador es la estimación de la posición y orientación de un objeto (de aquí en adelante pose de un objeto). El algoritmo POSIT (Pose from Orthography and Scaling with Iteration) es uno de los algoritmos clásicos a la hora de calcular la pose de un objeto conocido.
Para calcular la pose, POSIT necesita 4 puntos no coplanares del objeto, y el algoritmo asume que se encuentran a la misma profundidad. Esto es obviamente incorrecto, pero se asume que el objeto está lo suficientemente lejos como para que podamos prescindir del relieve del objeto. A este tipo de perspectivas se las denomina *weak-perspective*.
La ejecución del algoritmo se divide en dos partes. Primero, la fase POS se encarga de hallar una pose aproximada del objeto. Para ello utiliza los parámetros intrínsecos de la cámara, aquellos que calculamos al calibrar la cámara, para hallar la perspectiva del objeto, y por tanto podemos calcular la pose aproximada del objeto. Para hallar dicha perspectiva, el algoritmo necesita las coordenadas de los puntos en el sistema de coordenadas del objeto, y las coordenadas de los puntos en la imagen, y devuelve una matriz de rotación y un vector de escalado.
A continuación la fase POSIT, POS con Iteración, vuelve a calcular POS, pero en vez de utilizar como datos de entrada los puntos del objeto, utiliza los resultados de POS. De esta forma, retroalimentando el algoritmo, conseguimos mejorar los resultados, y en 4 o 5 iteraciones es capaz de converger a la pose real.
En OpenCV no llamamos a una función de forma iterativa, sino que marcamos un criterio de terminación, con el que se saldrá del bucle.

Objetos

En visión llamamos objetos a los objetos físicos que queremos modelar. Estos objetos suelen ser de un tamaño relativamente pequeño, por lo que es posible dar la vuelta a su alrededor para poder obtener imágenes de los mismos desde muchos puntos de vista.

Escenas

Llamamos escenas a las situaciones en las que el volumen a visualizar es demasiado grande, y los puntos de vista que podemos obtener de dicho volumen son limitados. Normalmente, en una escena no es posible dar la vuelta alrededor del volumen a visualizar, por lo que tan solo podremos generar resultados para los elementos que sean visibles desde un conjunto de vistas.
Es común que una escena esté compuesta por múltiples objetos.

Visión Monocular:

Conocemos con este nombre a los sistemas en los que se emplea un único punto de visión (ojo, cámara...) para obtener imágenes del entorno.
Debido a que solo tenemos una única imagen, en principio no podemos hallar la profundidad a la que se encuentran los objetos (ya que en principio son necesarias dos vistas de un mismo punto desde posiciones conocidas para estimar su profundidad), por lo que deberemos utilizar otras técnicas para poder generar información sobre profundidad.
La explicación sobre el uso de dos vistas de un mismo objeto para el cálculo de la profundidad será abordada en el apartado sobre Triangulación.

Técnicas monoculares:

Photometric Stereo:

Se conoce con este nombre a la técnica que utiliza los cambios en la iluminación para obtener los vectores normales de la superficie de un objeto 3D. Suele ser utilizada para la reconstrucción de objetos 3D. Para poder usar esta técnica, es necesario tener luces puntuales, de intensidad conocida y regulable (pueden ser encendidas y apagadas), y pose conocida. También se supone que la superficie del objeto a modelar es Lambertiana, esto es, que la luminosidad reflejada por el objeto sea uniforme, y que esta no tendrá *specular highlights*. No existen objetos con reflectividad Lambertiana en el mundo real, y dependiendo del material tendremos mayores o menores *specular highlights*, por lo que la precisión de este método está limitada por el tipo de objeto a modelar.

Para calcular la normal de una superficie con esta técnica se usa la siguiente ecuación:

$$I = n * L$$

Donde I es un vector de m intensidades observadas, n es la normal de la superficie y L es una matriz conocida de 3xm direcciones de luz normalizadas.

Para mantener la independencia de los vectores de L, y que por tanto el sistema sea resoluble, necesitamos que no haya dos luces en el mismo eje Azimut, el eje respecto a la vista del observador.

Como se puede ver, esta técnica puede ser utilizada tanto con una cámara, como con más, por lo que se ha optado por mencionarla en el apartado de visión monocular.

Shape from Focus:

Busca estimar la distancia del punto a partir del desenfoque, *blur*, del mismo. Los puntos que aparecen perfectamente enfocados son aquellos que se encuentran en el plano de foco, y cuanto más se alejan estos de dicho plano, más desenfocados están. Esta técnica tiene algunos problemas:

1. El desenfoque aumenta a medida que nos alejamos del plano, de forma proporcional y en **ambas** direcciones. Por tanto hay que encontrar un método para poder determinar a que lado del plano de foco se encuentra el punto cuya profundidad queremos estimar.
2. La estimación del grado de desenfoque no es fiable a día de hoy.

Texture Gradients:

Es una técnica similar a Shape from Focus. Detecta la distancia de los objetos en base a la distorsión de la textura de los mismos. Para detectar la distorsión, se centra en una propiedad de la textura, como su homogeneidad, isotropía..., y se compara la propiedad original de la textura, con la observada. Para poder llevar a cabo los cálculos, supone que la propiedad observada en la textura es uniforme, y que la iluminación no va a afectar a la textura (superficie Lambertiana). Esta técnica tiene algunos problemas:

1. La deformación de las texturas aumenta a medida que nos alejamos, pero también puede deberse a factores como iluminación, rugosidad, auto-oclusiones... No son fiables.
2. Los algoritmos generados hasta el momento solo tratan objetos rígidos lambertianos y con superficies regulares (The texture gradient equation for recovering shape from texture).

Funcionan en tres fases:

1. Identificar una propiedad de la textura, y estimar su estado original.
2. Calcular el gradiente de deformación de la textura.
3. Calcular las coordenadas 3D a partir del gradiente de deformación.

Active Range-finding:

Son métodos en los que se cambia el entorno de alguna forma para medir las reacciones del objeto. En el caso de *light stripe*, consiste en crear un plano de iluminación distinta en la escena y moverlo de tal forma que recorra el objeto. Esto puede conseguirse haciendo pasar una línea de luz (o láser) por el objeto, o generando una sombra sobre el mismo. Una vez se percibe el plano, se triangulan los puntos de corte del objeto con el plano de iluminación.

Reconstrucción densa

Ver 3-D Depth Reconstruction from a Single Still Image.

Visión Estéreo:

Llamamos así a los sistemas que emplean más de un sistema de visión (cámaras, ojos...) para obtener distintas imágenes del objeto o vista observado simultáneamente. Un ejemplo de este tipo de visión, serían los ojos de los seres humanos.

En este tipo de sistemas, se toman las imágenes obtenidas por las dos cámaras, y se buscan las correspondencias entre los puntos de las dos, esto es, se integra la información obtenida de . De esta forma, puede obtenerse información de profundidad a partir de las imágenes. Este punto se define en el apartado sobre Triangulación.

La búsqueda de correspondencias entre puntos de las imágenes es pesada, desde el punto de vista computacional, pero la carga de trabajo puede reducirse utilizando los datos geométricos con los que ya contamos, como por ejemplo la distancia entre las cámaras, y su orientación. Este punto se cubre en el apartado sobre geometría epipolar.

Los algoritmos de visión estéreo suelen utilizar 4 pasos:

1. Eliminación de aberraciones en las cámaras a utilizar. Esto se consigue mediante la calibración.
2. Ajustar los ángulos y distancias entre las cámaras, para conseguir que las imágenes generadas sean coplanares, y que las líneas de las imágenes estén perfectamente alineadas.
3. Buscar correspondencias entre las dos imágenes. Como resultado de este paso obtenemos un mapa de disparidades. Se conoce a este paso como correspondencia.
    1. Algunos métodos buscan puntos de interés, o puntos salientes en las imágenes, y por tanto generan mapas de disparidades únicamente para esos puntos. A esto se le llama emparejamiento discreto.
    2. Otros sistemas buscan correspondencias para todos los puntos de la imagen, y por tanto obtienen mapas de disparidad de toda la imagen. Se dice que estos sistemas generan emparejamientos densos.
    3. Sin duda el primer método es más rápido, pero para algunas aplicaciones la información que genera no es suficiente, por lo que debe usarse el emparejamiento denso.
4. Una vez tenemos el mapa de disparidades, podemos usarlo para generar triangulaciones, y obtener distancias a los puntos observados. A este paso se le llama “reprojección” y genera un mapa de profundidades, para puntos discretos, o para toda la imagen, dependiendo del tipo de emparejamiento que hayamos utilizado en el paso anterior.

Problemas de la visión estéreo:

Problema de correspondencia:

Conocemos con este nombre al problema de saber dado un punto p_1 en el plano de visión de la primera cámara, a que punto corresponderá en la segunda cámara.

Problema de reconstrucción:

Define el problema de obtener las coordenadas del espacio físico de un punto a partir de la información visual con la que contamos.

Problema de localización:

Para que muchos de los algoritmos de visión funcionen es necesaria información sobre la pose de diferentes elementos del sistema. Por ejemplo, si vamos a reconstruir en base a un sistema de coordenadas que no sea el de la cámara, necesitaremos conocer la pose de la cámara en ese sistema de coordenadas.
Dependiendo del algoritmo que usemos, es posible que sea necesario conocer la pose de más elementos del sistema, como luces, o una estimación de donde puede encontrarse el objeto o la escena a modelar.
Este problema será especialmente importante en el apartado de Visión Estéreo Multiview.

Visión estéreo mono-view

Conocemos con este nombre a los sistemas estéreo en el que con dos cámaras obtenemos dos imágenes simultáneamente para ser procesadas, y posteriormente no tenemos en cuenta ninguna otra imagen para el procesamiento. Suele ser utilizada para aplicaciones en las que la evaluación del contenido de un par de imágenes no depende del resto de pares que pueda obtener el sistema. También se la conoce como visión estéreo.

Visión estéreo multi-view

Llamamos así a los sistemas estéreo en los que procesamos más de un par de imágenes para obtener el resultado. Este conjunto de pares puede consistir en una secuencia de vídeo estéreo o un conjunto de pares sueltos, pero en todo caso, se utiliza más de un par para realizar el procesamiento de las imágenes, y el resultado depende del procesamiento de múltiples pares de imágenes.

Triangulación:

Se llama así al proceso de determinar la posición del punto P en el espacio físico, a partir de al menos dos imágenes distintas en las que se pueda ver el punto, y el conocimiento de la pose de las cámaras para cada una de las imágenes.

Supongamos que tenemos un sistema de visión con dos cámaras coplanares, esto es, que sus respectivos planos de imagen se encuentran dentro del mismo plano, que están separadas por una distancia conocida, cuyos ejes de visión son perpendiculares al plano donde se encuentran las cámaras, paralelos entre si, y cuentan con la misma dirección.
También vamos a suponer que la distancia focal y la amplitud del campo de visión de ambas cámaras es la misma, y es conocida.
Por último, consideramos que se han eliminado las aberraciones de las lentes de las cámaras, se han alineado las imágenes. Estos aspectos son cubiertos en profundidad en el apartado dedicado a la rectificación, y de aquí en adelante diremos que las imágenes obtenidas por ambas cámaras se encuentran rectificadas.

El punto P tendrá una pareja de coordenadas en ambas imágenes (x e y) y una de ellas será común en ambas imágenes, esto se debe a que las imágenes que estamos utilizando están rectificadas. Por el contrario la otra coordenada será distinta en cada imagen, la horizontal (x_l , x_r) en este proyecto, y es a partir de la diferencia entre estas coordenadas, también conocida como disparidad, como podemos obtener la profundidad del punto.
Si conocemos la distancia desde el plano de visión a la cámara, la distancia focal, podemos calcular el ángulo con el que partiría una recta que pase por P y por la cámara, y hacerlo para cada una de las cámaras. El punto en el que las dos rectas se crucen, encontraremos el punto P .
Una vez tenemos las dos rectas, para calcular la distancia desde P hasta el punto más cercano a la recta que une las dos cámaras solo tenemos que calcular la altura del triángulo formado por la línea que une las cámaras y las líneas que unen las cámaras con P . De este triángulo conocemos la base, y los tres ángulos, por lo que podemos hallar la altura del mismo (coordenada Z).

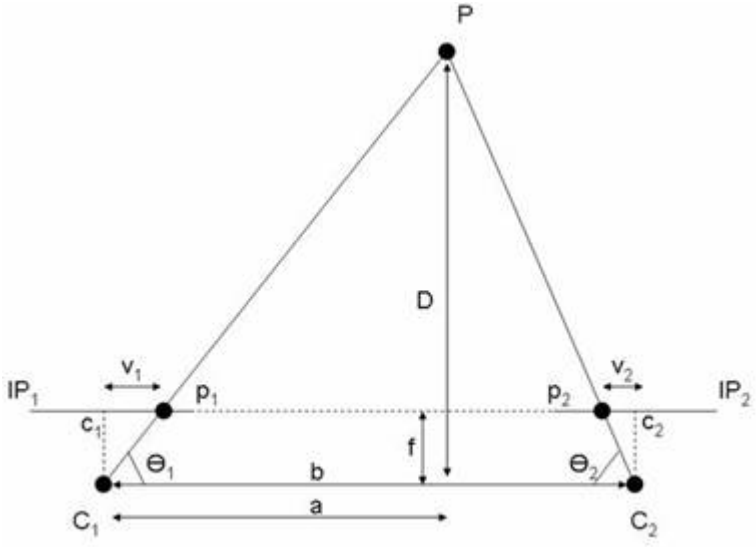

Ilustración 5: Triangulación en un sistema estéreo

Para calcular la distancia del objeto, vamos a tomar un sistema de dos cámaras, donde llamaremos B a la longitud de la recta que une las dos cámaras (baseline), D a la distancia entre el punto cuya posición queremos calcular (P) y el baseline. α y β serán los ángulos entre B y las rectas unen las cámaras y el punto. Por último llamaremos f a la distancia focal de las cámaras (que estamos suponiendo que es igual para ambas cámaras). De esta forma tenemos:

$$D = \frac{B \sin(\alpha) \sin(\beta)}{\sin(\alpha + \beta)} - f$$

Otra forma de calcular la distancia D puede ser la siguiente:

$$D = f\left(\frac{B}{x_l + x_r} - 1\right)$$

Donde x_l y x_r son las distancias horizontales que separan en el plano de visión el eje óptico y el punto donde corta el plano de visión el rayo que une la cámara y el punto P cuya posición queremos calcular.

Solo en el caso de que queramos mejorar la medición de distancias de objetos muy cercanos al sistema de visión, rotaremos las cámaras de tal forma que sus ejes de visión se crucen en un espacio finito.
También es importante sincronizar las cámaras, de lo contrario no obtendremos posiciones correctas en objetos en movimiento, ya que cada cámara registrará la imagen del objeto móvil en momentos distintos.

Debido a que el cálculo de la distancia depende de los ángulos α y β y de la distancia entre cámaras B , sus valores y la sensibilidad de nuestro sistema frente a las variaciones en los mismos influirán en el grado de corrección de la distancia calculada.
La sensibilidad en la percepción de los ángulos es algo inherente a los sensores que estemos utilizando, y no se podrá modificar, salvo que cambiemos los sensores.
El parámetro que podemos modificar es B . Si bien la distancia entre los sensores puede ser fija, podemos mover el sistema de visión, y de esta forma obtenemos imágenes más espaciadas entre si. Este punto se desarrolla en el apartado sobre Structure From Motion.

Triangulación monocular utilizando múltiples imágenes:

Para poder estimar la profundidad de un punto de la geometría física mediante triangulación es necesario tener 2 vistas de dicho punto, y que conozcamos la geometría que relaciona las vistas entre si. En el apartado sobre visión monocular nos hemos centrado en el cálculo de la profundidad a partir de una única imagen, pero podemos tomar múltiples imágenes con una única cámara en movimiento que genere al menos dos vistas distintas del punto que queremos calcular. Siempre y cuando conozcamos el cambio de posición de la cámara (**R**otación y **T**raslación) entre una imagen y otra podremos calcular la profundidad a la que se encuentra P .
Esta relación entre las imágenes nos vendrá dada por la matriz **E**sencial (ver apartado Cálculo de la matriz Esencial, E).

Triangulación en el proyecto:

En este proyecto primero se busca la relación entre las cámaras del sistema estéreo, y a continuación se genera un mapa de disparidades. En este mapa se representan las diferencias de coordenadas para un punto entre las dos imágenes, y se cubre en mayor profundidad en el apartado de Correspondencia Estéreo. A continuación se utiliza el método cvReprojectImageTo3D de OpenCV para calcular la posición física de los puntos presentes en el mapa de disparidades. Este método toma como origen de coordenadas la posición de la primera cámara, en nuestro caso la cámara izquierda. En realidad el origen de coordenadas utilizado en el método depende de las coordenadas suministradas en el mapa de disparidades.

Geometría Epipolar:

A la hora de usar sistemas estéreo utilizamos una geometría especial llamada geometría epipolar. Para explicar lo que es, primero debemos centrarnos en los puntos epipolares, e_l y e_r de aquí en adelante.
Primero vamos a tomar dos cámaras coplanares y sus respectivos planos de imagen. Se va a trazar una recta desde cada cámara hasta el punto observado, P, y se va a denominar p_l y p_r a los puntos en los que estas rectas corten los planos de imagen. Vamos a unir las cámaras mediante una línea imaginaria, y vamos a llamar e_l al punto en el que dicha recta corta el plano de imagen de la primera cámara, y e_r a su contrapartida. Dicho en otras palabras, los puntos epipolares son la proyección en el plano de imagen de la otra cámara.
Al plano formado por los puntos epipolares y el punto observado se le llama plano epipolar, y a la línea en la que el plano epipolar corta el plano de imagen se le llama línea epipolar.
Una vez explicado esto, se van a enunciar algunas propiedades importantes de este sistema:

1. Todos los puntos 3D que pueden ser vistos por una cámara están contenidos dentro de un plano epipolar, que a su vez intersecta el plano de imagen creando una línea epipolar.
2. Dado la vista de un punto 3D en un plano de imagen, la vista de dicho punto en el otro plano de imagen debe de estar dentro de la recta epipolar. A esto se le conoce como la restricción epipolar.
3. Se mantiene el orden de los puntos. Si los puntos A, B y C aparecen en ese orden en el plano de imagen de una cámara, aparecerán en dicho orden en la otra, salvo que una oclusión impida ver alguno de los mismos.

Por tanto si sabemos que el punto que buscamos está dentro de una recta, no tenemos que hacer una búsqueda en toda la imagen a la hora de buscar correspondencias entre puntos de las dos imágenes, lo cual reduce el problema de la correspondencia en una dimensión (de 2 a 1).

Calibración Estéreo:

Llamamos así al proceso de calcular la relación geométrica entre dos cámaras en el espacio. Para representar la relación entre las cámaras, utilizamos la matriz de **R**otación y el vector de **T**raslación. Por tanto, el objetivo del proceso de calibrado estéreo es el cálculo de R y T. Para calcular estos parámetros, debemos empezar por calcular la posición de un mismo punto P en ambas cámaras. El proceso para identificar el mismo punto en las dos cámaras será abordado en el apartado dedicado a la correspondencia. Vamos a tomar P como las coordenadas en el espacio físico del punto que queremos localizar, y P_l, P_r las coordenadas físicas del punto en los sistemas de referencia de cada una de las cámaras, esto es, en un sistema de coordenadas donde el origen se encuentre en la cámara. Podemos mover P al sistema de coordenadas de una cámara utilizando $P_l = R_l P + T_l$. También sabemos que las dos vistas están relacionadas, de forma que $P_l = R^T(P_r - T)$. De estas ecuaciones se obtiene:

$$R = R_r (R_l)^T$$
$$T = T_r - R T_l$$

Con estas ecuaciones estamos obteniendo unos parámetros R y T que permiten pasar puntos del sistema de referencia de una cámara al de la otra. En concreto, OpenCV traslada ambos sistemas de referencia a un tercero intermedio, de tal forma que minimiza las transformaciones que se deben efectuar, y minimiza el error de reproyección, o la diferencia entre la posición estimada del punto y la real. Para calcular estos parámetros en OpenCV utilizamos la función cvStereoCalibrate(..).
En el caso de cvStereoCalibrate() se buscan primero las coordenadas de un punto del plano físico relativas a cada cámara, de tal forma que obtenemos R_l , R_r , T_l y T_r. A continuación se generan R y T. La formula utilizada para obtener estos parámetros es:

$$R_r = R * R_l \quad T_r = R * T_l + T$$

Calibración Estéreo en el proyecto:

En el proyecto se realiza la calibración de las cámaras, hallar los parámetros intrínsecos y extrínsecos de las mismas, a la vez que la calibración estéreo. De esta forma, tomamos una única vez los puntos de calibración, del tablero de ajedrez, y dado que podemos ver los mismos puntos en las dos imágenes, los utilizamos para el cálculo de la **R**otación y **T**raslación de las cámaras.
Para hacerlo utilizamos la función incluida dentro de OpenCV cvStereoCalibrate. Esta función, sin embargo está limitada a hallar R y T en sistemas en los que la alineación de cámaras sea horizontal o vertical, y en el que las cámaras sean coplanares, por lo que no puede ser utilizado en sistemas de Estéreo Multiview. Este punto es explicado en mayor profundidad en el apartado sobre Structure From Motion y en el subapartado Bundle Adjustment.

Matrices Esenciales y Fundamentales:

De aquí en adelante utilizaremos matriz E para referirnos a la matriz esencial y F a la fundamental. La matriz E contiene la relación física entre las dos cámaras, en concreto la matriz de rotación y el vector de traslación. F además de la información contenida en E contiene las matrices de cámara M de ambas cámaras.
La matriz E nos permite tomar un píxel de la imagen 1, y proyectar una línea en la imagen 2 en la cual se debe de encontrar la proyección de dicho punto. Además sabemos que esta línea debe pasar por el punto epipolar [6][14].

Cálculo de la matriz Esencial, E:

Dado el punto del espacio físico P , vamos a llamar P_l y P_r a las coordenadas del mundo físico que ese punto tendrá en el sistema de coordenadas de la cámara 1 (O_l) y la cámara 2(O_r). Es importante tener en cuenta que estamos suponiendo que ambas cámaras están normalizadas, y que por tanto no necesitamos las matrices de cámara M para proyectar los puntos al espacio físico.
Nos interesa relacionar P_l y P_r , o dicho de otra forma, hallar la correspondencia del punto P_l para el sistema de coordenadas de la cámara O_r . Tal y como se ha explicado anteriormente, la conversión entre un sistema de coordenadas y otro se realiza mediante la matriz R y el vector T , en concreto utilizando la fórmula:

$$P_r = R(P_l - T)$$

Tal y como habíamos explicado en el apartado anterior, gracias a la geometría epipolar sabemos que el punto P_r no puede estar en cualquier punto del espacio, sino que tiene que ser parte del plano epipolar. Es importante resaltar que debido a que estamos situando puntos en el espacio físico, 3 dimensiones, las restricciones epipolares reducen el ámbito de búsqueda del punto a un plano, y no a una línea, como pasaba a la hora de buscar correspondencias. Por lo que si llamamos **x** al conjunto de puntos de un plano que tenga un vector normal **n** y pase por el punto **a** tenemos:

$$(x - a) * n = 0$$

También sabemos que el plano epipolar contiene los vectores P_l y T , y que si queremos crear un vector perpendicular al plano solo tendríamos que multiplicar ambos vectores $(T * P_l)$. Si sustituimos en la ecuación anterior:

$$(P_l - T)^T (T * P_l) = 0$$

Partiendo de $P_r = R(P_l - T)$ podemos reescribir la formula como $(P_l - T) = R^{-1} * P_r$. Si sustituimos $R^{-1} = R^T$ obtenemos:

$$(R^T P_r)^T (T * P_l) = 0$$

Vamos a redefinir $(T * P_l)$ como:

$$T * P_l = SP_l \Rightarrow S = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix}$$

Tomando todo lo definido anteriormente, definimos la función con la que calculamos E:

$$(P_r)^T RSP_l = 0$$
$$P_r^T EP_l = 0$$

También podemos expresarla como la siguiente fórmula, donde T es la matriz de translación en formato 3x3 y R la matriz de rotación:

$$E = RT = R \begin{bmatrix} 0 & -T_2 & T_1 \\ T_2 & 0 & -T_0 \\ -T_1 & T_0 & 0 \end{bmatrix}$$

Cálculo de la matriz fundamental, F:

Dado el punto observado P, vamos a llamar Q_l y Q_r a las coordenadas de los píxeles de los planos de imagen en los que se ve el punto en las cámaras O_l y O_r respectivamente. La relación entre dichos puntos viene dada por la matriz F.
Siendo M la matriz de parámetros intrínsecos calculada durante la calibración, tenemos que:

$$Q = MP \Rightarrow P = M^{-1}Q$$

Si sustituimos las P en la ecuación en la que calculábamos E obtenemos:

$$Q_r^T (M_r^{-1})^T E M_l^{-1} Q_l = 0$$

También sabemos que F es:

$$F = (M_r^{-1})^T E M_l^{-1}$$

Con lo que podemos limpiar la ecuación anterior, y obtenemos la ecuación con la que podemos calcular F:

$$Q_r^T F Q_l = 0$$

En resumen, las funciones para calcular E y F son muy similares. Lo único que cambia entre ellas es el tipo de puntos utilizados para el cálculo. En E son coordenadas físicas, y en F coordenadas de píxeles.

Cálculo de las matrices E y F en OpenCV:

Tan solo tenemos que proporcionar al sistema un conjunto de puntos con sus correspondencias en la otra cámara, y la función cvFindFundamentalMat(..) nos devolverá F. Para poder calcular la matriz es necesario que proporcionemos al sistema al menos 7 puntos con sus correspondencias en la otra cámara. Debido a que los puntos suministrados pueden no generar la información suficiente para calcular F, es importante comprobar que el valor de retorno de la función no es 0. Si lo es, no ha podido calcular la matriz F.
Las matrices E y F también son resultados devueltos por la función cvStereoCalibrate. Cuando se calculan mediante esta función se utilizan las siguientes ecuaciones:

$$E = \begin{bmatrix} 0 & -T_2 & T_1 \\ T_2 & 0 & -T_0 \\ -T_1 & T_0 & 0 \end{bmatrix} R$$

$$F = (M_r)^t E (M_l)^t$$

Donde M_l es la matriz de la cámara izquierda, y M_r la de la cámara derecha, y $T = (T_0, T_1, T_2)$, esto es, son las componentes **x**, **y**, **z** de T.

Matrices E y F en el proyecto:

En el proyecto apenas se utilizan las matrices E y F como tales, sino que se ha optado por utilizar sus componentes, R y T. Se calculan inicialmente durante la calibración del sistema estéreo, pero luego no vuelven a utilizarse.

Rectificación Estéreo:

Con la calibración hemos conseguido situar los planos de imagen de ambas cámaras en el mismo plano, pero a pesar de ello, es posible (y muy probable) que a pesar de estar en el mismo plano, las imágenes que obtengan las cámaras no estén alineadas. El objetivo del proceso de rectificación es alinear las imágenes, de tal forma que la búsqueda de un píxel en un plano de imagen se reduzca a la búsqueda en una recta. Para ello, este proceso va a necesitar las matrices generadas en el paso de calibración, esto es R y T.
OpenCV utiliza dos algoritmos, el de Hartley, y el de Bouguet. El resultado de ambos algoritmos es una matriz de rotación llamada R_{rect} que alinea las imágenes. En este proyecto se ha optado por utilizar el algoritmo de Bouguet, ya que se observó que producía mejores resultados que el algoritmo de Hartley.

Algoritmo de Bouguet:
Dadas las matrices R y T de un sistema estéreo, el algoritmo intenta reducir al mínimo el cambio producido en las imágenes (para reducir distorsiones), aumentando al máximo el área de visión común.
Para reducir al mínimo las distorsiones en la imagen al producirse la reproyección, no se lleva el plano de imagen derecho al sistema de coordenadas del izquierdo, sino que se lleva ambos planos de imagen (izq y dcho) a un plano intermedio, de tal forma que las distorsiones causadas por este cambio sean las menores posibles. Para hacer esto, se divide en dos la matriz R , y se obtienen las matrices de rotación r_l y r_r .
El aplicar estas rotaciones sitúa los planos de imagen en el mismo plano, y consigue que los ejes de visión sean paralelos, pero no alinea las imágenes en si. Para poder alinear las imágenes, necesitamos calcular R_{rect} , que es una rotación en torno al eje de visión de los dos planos de visión.
Dado que el vector de **T**ranslación relaciona el punto C (punto donde el eje de visión corta el plano de visión) de un plano de visión con el punto C del otro plano, vamos a rotar ambos planos de imagen de tal forma que sean paralelos horizontalmente con el mismo.

Vamos a dividir el cálculo de R_{rect} en el cálculo de 3 vectores unitarios. Designamos el primero como el vector unitario del vector de traslación:

$$e_1 = \frac{T}{\|T\|}$$

El siguiente componente puede ser cualquier vector ortogonal al anterior, pero vamos a hacer que también sea ortogonal al eje de visión.

$$e_2 = \frac{[-T_y \; T_x \; 0]^T}{\sqrt{T_x^2 + T_y^2}}$$

El último componente debe de ser ortogonal a los dos anteriores:

$$e_3 = e_1 \, x \, e_2$$

Generamos R_{rect} :

$$R_{rect} = \begin{bmatrix} (e_1)^T \\ (e_2)^T \\ (e_3)^T \end{bmatrix}$$

Por tanto, a la hora de calibrar y rectificar imágenes estéreo, utilizaremos las siguientes matrices para la imagen izquierda y derecha respectivamente:

$$R_l = R_{rect} r_l$$
$$R_r = R_{rect} r_r$$

De esta forma colocamos los dos planos de visión en el mismo plano y los rectificamos con una única multiplicación matricial.

También nos interesa obtener las matrices de proyección, que nos permiten convertir los puntos del espacio físico en puntos del plano de imagen normalizado. Esta matriz se obtiene de:

$$P_i = M_{i_{rect}} [R_i | T_i]$$

Debido a que el sistema de cámaras que vamos a utilizar para este proyecto se encuentra prácticamente en el mismo plano, y la diferencia en sus encaramientos es mínima, sabemos que la matriz de rotación va a tender a ser muy similar a la matriz identidad, por lo que en los siguientes ejemplos la vamos a sustituir la rotación por la matriz identidad.

$$P_l = M_{rect\,l} P_l' = \begin{bmatrix} f_{x_l} & \alpha_l & c_{x_l} \\ 0 & f_{y_l} & c_{y_l} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & T_{x_l} \\ 0 & 1 & 0 & T_{y_l} \\ 0 & 0 & 1 & T_{z_l} \end{bmatrix} = \begin{bmatrix} f_{x_l} & 0 & c_{x_l} & 0 \\ 0 & f_{y_l} & c_{y_l} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$P_r = M_{rect\,r} P_r' = \begin{bmatrix} f_{x_r} & \alpha_r & c_{x_r} \\ 0 & f_{y_r} & c_{y_l} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & T_{x_r} \\ 0 & 1 & 0 & T_{y_r} \\ 0 & 0 & 1 & T_{z_r} \end{bmatrix} = \begin{bmatrix} f_{x_r} & 0 & c_{x_r} & T_x * f_x \\ 0 & f_{y_r} & c_{y_r} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

En este caso, estamos suponiendo que las cámaras se encuentran en a la misma altura, esto es, que tenemos que realizar una rectificación horizontal. Además, hemos decidido que será una de las cámaras la que será trasladada (la derecha), y por ello sabemos que solo T_{x_r} tiene un valor distinto de 0. En el caso de que fuésemos a realizar una rectificación en el eje y utilizaríamos:

$$P_l = \begin{bmatrix} f_{x_l} & 0 & c_{x_l} & 0 \\ 0 & f_{y_l} & c_{y_l} & T_y * f_{y_l} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

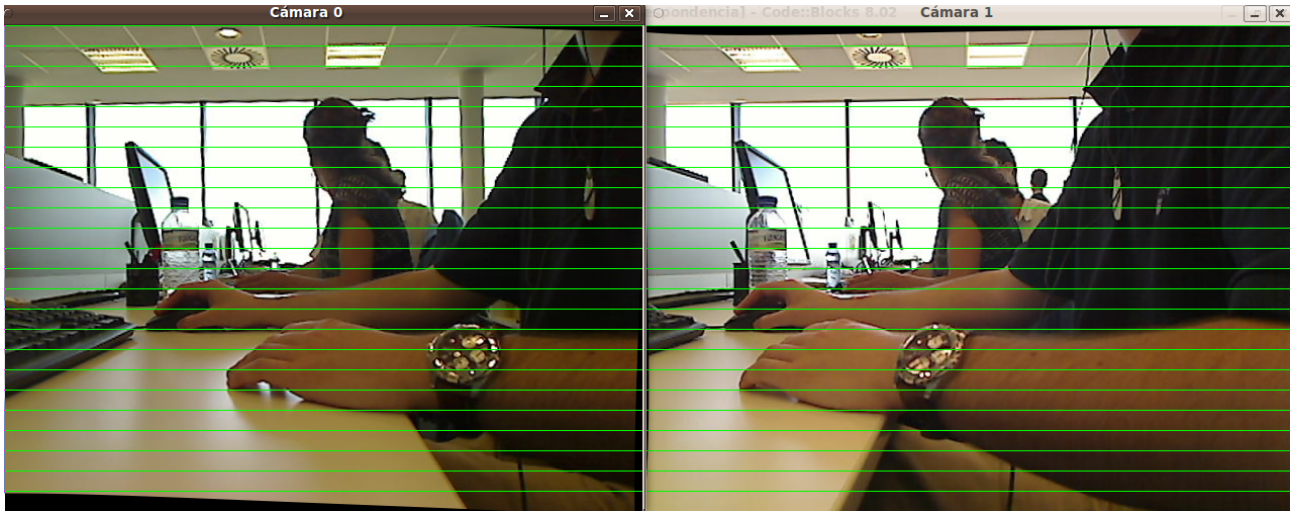

Ilustración 6: Ejemplo de rectificación estéreo

Para este ejemplo, en el que ambas cámaras se encuentran a unos 11cm de distancia, se han generado las siguientes matrices de rotación utilizando el método de Bouguet:

$$Rl = \begin{bmatrix} 0.99 & -0.04 & -0.005 \\ 0.04 & 0.99 & -0.0009 \\ 0.005 & 0.0007 & 0.99 \end{bmatrix} Rr = \begin{bmatrix} 0.99 & 0.01 & -0.003 \\ -0.01 & 0.99 & 0.0009 \\ 0.003 & -0.0008 & 0.99 \end{bmatrix}$$

Como se puede comprobar, el valor de las matrices tiende a la matriz identidad, debido a que las dos cámaras se encuentran en el mismo plano.

$$Pl = \begin{bmatrix} 584.59 & 0 & 322.73 & 0 \\ 0 & 584.59 & 235.72 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} Pr = \begin{bmatrix} 584.59 & 0 & 322.73 & -2224.32 \\ 0 & 584.59 & 235.72 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

De la rectificación de Bouguet también obtenemos otro parámetro, la matriz de reproyección Q, que relaciona las coordenadas de píxel con puntos del espacio físico en un sistema rectificado, permitiéndonos reproyectar los puntos de la imagen a puntos del espacio físico, esto es, lo contrario que la matriz de proyección P . Sus componentes son los siguientes:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T_x & (c_x - c'_x)/T_x \end{bmatrix}$$

Donde c_x, c_y son las coordenadas en píxeles del punto principal de la cámara izquierda, f es la distancia focal de la cámara izquierda expresada en píxeles, T_x es la translación en el eje x (dado que estamos haciendo una rectificación horizontal no realizamos ninguna traslación en el eje y) y c'_x es la coordenada x del punto principal de la cámara derecha, también expresado en píxeles.

El motivo por el que estamos utilizando las coordenadas de la cámara izquierda, es que al rectificar la imagen hemos trasladado las posiciones de los píxeles al sistema de coordenadas de la cámara izquierda.

Esta matriz será luego utilizada para calcular las coordenadas de puntos físicos a partir de coordenadas de imagen, y es cubierto en el punto dedicado a Mapas de Profundidad.

Eliminando las distorsiones:

El algoritmo de Bouguet solo va a calcular las matrices P y R que hay que aplicar a los planos de visión, pero a continuación hay que realizar dichas modificaciones a los planos. Para ello, OpenCV utiliza dos funciones distintas, “cvInitUndistortRectifyMap” y “cvRemap”.

La función cvInitUndistortRectifyMap va a utilizar las matrices P y R para calcular la rectificación, pero además, va a recibir la matriz M y los coeficientes de distorsión para poder eliminar las distorsiones. Los parámetros que va a recibir la función son por tanto:

$$M=\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \qquad P=\begin{bmatrix} f'_x & 0 & c'_x \\ 0 & f'_y & c'_y \\ 0 & 0 & 1 \end{bmatrix}$$

Por tanto, esta función calcula las modificaciones que se deben realizar al plano de imagen de una cámara, para rectificar su imagen y eliminar las distorsiones causadas por el hardware de la cámara. Estas modificaciones se presentan en forma de dos mapas, **x** e **y**. Primero, se calculan las modificaciones que se deben llevar a cabo para rectificar la imagen, y a continuación, se vuelve a modificar este resultado para aplicar las modificaciones de los coeficientes de distorsión.

Para cada punto de la nueva imagen, des-distorsionada y rectificada, vamos a querer calcular cual es el píxel de la imagen original que debe trasladarse allí. Para hallarlo vamos a utilizar las siguientes ecuaciones:

$$x=\frac{u-c'_x}{f'_x} \qquad y=\frac{v-c'_y}{f'_y}$$

$$[XYW]^T \Leftarrow R^{-1}*[xy1]^T$$

$$x'=\frac{X}{W} \qquad y'=\frac{Y}{W}$$

Con estas ecuaciones calculamos la posición del píxel que debemos trasladar a las coordenadas u,v para rectificar la imagen. Ahora lo que queda es des-distorsionarla.

$$x''\Leftarrow x'(1+k_1r^2+k_2r^4+k_3r^6)+2\mathrm{p}_1x'y'+p_2(r^2+2\mathrm{x}'^2)$$

$$y''\Leftarrow y'(1+k_1r^2+k_2r^4+k_3r^6)+p_1(r^2+2\mathrm{y}'^2)+2\mathrm{p}_2x'y'$$

$$map_x(u,v)\Leftarrow x''f_x+c_x \qquad map_y(u,v)\Leftarrow y''f_y+c_y$$

Con esto obtenemos dos mapas de transformaciones, verticales y horizontales a realizar al plano de imagen, para poder rectificar y des-distorsionar las imágenes tomadas por esa cámara.

Una vez se tienen los mapas, hay que utilizarlos en cada frame, para rectificar y des-distorsionar la imagen capturada. Para hacer esto se llama a la función cvRemap. Esta función recibe como parámetros la imagen capturada, los mapas calculados por cvInitUndistortRectifyMap, y por último, opcionalmente, un flag y un valor. La función utiliza los mapas para retocar la imagen, y devuelve la imagen modificada.

Es posible que haya algunas zonas del mapa (u,v) cuyos píxeles de origen se encuentren fuera del campo de visión de la cámara, y en ese caso, esos píxeles quedan vacíos, tal y como puede verse en la ilustración nº 6. Para evitar esto, es posible pasarle a cvRemap un flag, en el que se le indica que debe rellenar estos píxeles con un valor determinado (los dos parámetros opcionales).

Correspondencia Estéreo:

Llamamos así al emparejamiento de las proyecciones de un punto del espacio físico en las imágenes percibidas en las cámaras. Como es lógico, solo podremos identificar dichas proyecciones si ambas imágenes contienen el mismo punto, por tanto el área para la que podemos calcular la correspondencia es aquella visible desde las dos cámaras.
Una vez conocemos las posiciones relativas de las cámaras (matriz E, o matriz R y vector T), podemos calcular la disparidad de los puntos con correspondencia, utilizando la siguiente función:

$$d = x^l - x^r$$

Llamamos disparidad d a la diferencia que existe entre las coordenadas x^l, x^r en las que se visualiza un mismo punto en las dos cámaras. La disparidad será mayor cuanto más cerca esté el objeto de la cámara, y menor cuanto más alejado se encuentre. Para este caso estamos suponiendo que la imagen se encuentra rectificada en el eje Y, y que por tanto la coordenada Y de ambos píxeles es la misma.
En el caso de que los ejes de visión de las dos cámaras intersecten en espacio finito, que no sean paralelos, tenemos que utilizar esta otra fórmula:

$$d = x^l - x^r - (c_x^l - c_x^r)$$

Donde c son las coordenadas en el eje x del punto donde el eje de visión corta el plano de visión.
Como resultado del cálculo de correspondencias obtenemos un mapa de disparidad, una imagen en gama de grises, en la que se representan los para los que se ha encontrado correspondencia en la otra imagen. La tonalidad de gris del pixel en el mapa de disparidades depende del grado de disparidad que presenta el punto, y dado que la disparidad depende de la distancia a la que se encuentra el punto del espacio físico del sistema de cámaras, utilizaremos este mapa para obtener la coordenada Z de lo observado.

En este proyecto a la hora de buscar correspondencias entre las dos imágenes se han utilizado 3 tipos de algoritmos, BM, GC y SGBM. Estos algoritmos, que serán tratados en detalle más adelante, intentan obtener un mapa de disparidades que cubra la totalidad del área visible desde las dos cámaras, si bien no todos ellos son capaces de obtener valores de disparidad para todos los píxeles. Decimos que los mapas de disparidad generados por estos algoritmos son densos.
Existen aplicaciones de visión artificial en las que no es necesario calcular la correspondencia de todos los puntos, y nos baste con relacionar un conjunto de puntos entre las dos imágenes. En este caso decimos que estamos obteniendo un mapa de correspondencias discretas. Un ejemplo de este tipo de aplicaciones puede ser el proceso de calibración de las cámaras, en la que solo buscamos correspondencias para las intersecciones de las celdas del tablero de ajedrez.

Etapas del proceso de correspondencia en OpenCV:

Para explicar los pasos seguidos por los algoritmos de correspondencia se va a tomar como ejemplo la implementación en OpenCV del algoritmo BM.

La ejecución del algoritmo BM implementado en OpenCV puede dividirse en los siguientes tres pasos:

1. Prefiltrado para normalizar o intensificar la intensidad de la imagen.
2. Se buscan correspondencias de forma paralela a las líneas epipolares.
3. Se realiza un filtrado de los resultados obtenidos para eliminar falsos positivos.

A la hora de comparar píxeles entre si para detectar correspondencias BM utiliza el valor de intensidad de los píxeles evaluados y el de sus vecinos. Al conjunto formado por píxel y vecinos le llamamos ventana. De aquí en adelante cuando hablemos de ventanas nos referiremos a un píxel y a un conjunto de píxeles vecinos.

Prefiltrado:
Durante el prefiltrado puede normalizarse la imagen o intensificarse los bordes. Si se normaliza la intensidad de la imagen, se obtiene la intensidad de los píxeles restándole la media de la intensidad de los píxeles de su ventana, y estableciendo unos valores máximos y mínimos que el usuario puede definir, pero que por defecto son de 30 y -30. Para esto se usan ventanas de 5 píxeles de lado como mínimo (el tamaño de ventana de prefiltrado es independiente del tamaño de ventana del paso de búsqueda).

$$I = min[max(I_c - \bar{I}, -I_{cap}), I_{cap}]$$

Donde I_{cap} tiene un valor por defecto de 30 y $\bar{I}$ representa la media de la intensidad de todos los píxeles de la ventana. Este tipo de filtros suelen ser utilizados para reducir el ruido de las imágenes.

En el caso de querer intensificar los bordes se realiza un filtro utilizando un kernel gaussiano, esto es, hacemos que los píxeles centrales de la ventana tengan un peso muy superior a los externos a la hora de calcular la intensidad final de los píxeles. Con esto se consigue resaltar los bordes de los objetos presentes en la imagen.

Búsqueda:
A la hora de realizar la búsqueda, definimos un tamaño de ventana, y buscamos a lo largo de las líneas epipolares correspondencias entre ventanas. Dicho de otra forma, tomamos una ventana en la primera imagen, y para cada píxel de la la línea epipolar de la otra imagen tomamos una ventana de igual tamaño que la original y las comparamos. Para comparar las ventanas entre si, se toman los valores de intensidad de los píxeles que componen las ventanas y se suman entre si. A continuación se resta el valor de la ventana original a la de la nueva ventana, y se guarda el valor absoluto de la resta. Tras procesar todos los píxeles de la línea epipolar, se toma como correspondencia la ventana con el valor más cercano a 0. A este algoritmo de evaluación se le conoce como SAD, Sum of Absolute Differences.
Con el nombre de *horopter* conocemos al volumen de espacio físico en el cual buscamos correspondencias. Este volumen viene definido por los parámetros límite mínimo de disparidad, y número de disparidades (número de píxeles en los que buscar correspondencias). Los valores de disparidad altos suelen representar puntos cercanos, mientras que los de baja disparidad representan los lejanos. Dependiendo de los valores que indiquemos para estos parámetros, es posible que nuestro algoritmo no sea capaz de detectar profundidades para todos los puntos, ya que puede haber puntos físicos fuera del volumen en el que estemos buscando. Nos interesa mantener el volumen del *horopter* lo más ajustado posible a nuestras necesidades, ya que a mayor volumen, mayores serán las necesidades de computación. De esta forma podemos reducir la cantidad de comparaciones a realizar por cada píxel.
Una forma de mejorar este algoritmo es tener en cuenta que los puntos de la imagen izquierda, siempre aparecerán en el mismo orden en la imagen derecha, si bien es posible que alguna de estos puntos no sea visible. A esto se le llama "*order constraint*" [6].

La fórmula que representa el nivel de detalle en profundidad que podemos obtener es la siguiente:

$$\Delta Z = \frac{Z^2}{fT} \Delta d$$

Donde Z es el menor nivel de detalle que podemos obtener, d la menor disparidad que permitimos y T es la distancia entre las cámaras o *baseline*.

Postprocesado:
Los resultados de una función de disimilaridad rara vez indican un único posible resultado, y a la hora de elegir entre los posibles resultados elegimos aquel que tenga un mayor parecido con el bloque original. Esto puede llevar a falsos positivos, y para prevenirlo este paso filtra los resultados que sean “peores” que un cierto valor (parámetro *uniqueness Ratio*).

También comprobamos que la ventana utilizada en la búsqueda cuenta con un nivel de textura, una variedad de imagen, lo suficientemente grande como para poder superar los errores introducidos por el ruido. Esto viene reflejado por el parámetro *texture Threshold*.

Los bordes de los objetos son siempre problemáticos en visión estéreo, debido a que estamos viendo el objeto desde diferentes ángulos, los píxeles que correspondan al background van a ser distintos en las dos imágenes. A las regiones que marcan los bordes de los objetos, se las conoce como *speckle*, y para mejorar la detección de correspondencias utilizamos detectores especializados. Estos detectores utilizan también un sistema de ventanas, y se buscan disimilaridades, pero siempre y cuando los resultados devueltos se encuentren dentro de un rango, daremos por buenos los resultados.

Tipos de procesado de disparidades:

Correspondencia discreta: SIFT (Scale Invariant Feature-Transform)
Este algoritmo, definido en [20] permite detectar puntos relevantes o *features* en una imagen. Para cada punto, guarda además de su posición en la imagen información sobre los píxeles que lo rodean, de tal forma, que si se toman varias imágenes del mismo objeto o escena, sea posible reconocer el mismo punto.
Los identificadores de features obtenidos mediante este algoritmo no se ven afectados por rotaciones, translaciones o escalados, y es parcialmente resistente a cambios de iluminación. Todo esto facilita la correspondencia, ya que reduce la posibilidad de que surjan falsos positivos.
Se utiliza en reconstrucción para representar los puntos de la nube de puntos, y para poder establecer correlaciones entre puntos entre las diferentes imágenes.
No puede usarse para reconstrucción densa debido a que solo aceptará como feat puntos de alto contraste, por lo que la cantidad de puntos que representarán una superficie depende únicamente de las diferencias de contraste presentes en la misma.
Es un algoritmo patentado, y en EEUU es necesario pagar por su uso.
En el proyecto se utiliza para la búsqueda de correspondencias entre entre imágenes capturadas en distintos momentos. Para más información ver el apartado sobre Bundle Adjustment.

Correspondencia discreta: SURF (Speeded Up Robust Features)
Definido en [21].Se lo considera una variante de SIFT. Presentado en 2006, permite detectar más puntos que SIFT, procesarlos más rápido y de forma más robusta.
Los autores permiten un uso libre del mismo siempre y cuando este sea no comercial, y proporcionan gratuitamente librerías con la implementación del mismo.
En la actualidad se está realizando una implementación del mismo en la librería OpenCV.

Correspondencia discreta: *Patch*
Un *patch*[22]es un plano rectangular tangente a la superficie visualizada en su centro, cuyo vector normal está orientado hacia la cámara desde la que se observa.
Se utilizan múltiples patches para describir una nube de áreas en la superficie del objeto a reconstruir. El área del rectángulo del patch permite visualizar un área de la imagen de $\mu x \mu$ píxeles, pero solo tiene información geométrica para su punto central, usándose el resto de la información visual para localizarlo entre imágenes.

Correspondencia densa local, Block Matching:
Llamamos algoritmos locales a aquellos que solo buscan correspondencias para cada píxel y establecen la equivalencia de los puntos basándose en la similaridad de los píxeles vecinos contenidos dentro de una ventana. Además, la métrica utilizada para determinar la similaridad entre dos ventanas es una métrica que compara las intensidades de los píxeles de la ventana.
Como ejemplo de este tipo de algoritmos vamos a tomar Block Matching (BM), y vamos a analizar su implementación en OpenCV.
Los algoritmos de *block matching* son bastante rápidos, permitiendo comparaciones casi en tiempo real (más de un *frame* por segundo en nuestra implementación). Sin embargo, no siempre cuenta con una textura lo suficientemente diferenciada de su entorno, o con suficiente variabilidad de intensidad entre los píxeles de la ventana, por lo que solo se obtienen correspondencias para parte de los píxeles de la imagen. El segundo problema con el que nos encontramos con estos algoritmos aparece en los píxeles situados cerca del borde. Debido a que sus ventanas de correspondencia incluirán píxeles de profundidades diferentes, distintos para diferentes posiciones de la cámara, tendremos valores de intensidad muy distintos para las ventanas de ambas imágenes.
Para solucionar estos dos problemas, se suelen utilizar ventanas de tamaño variable[23], o se da una mayor o menor importancia, peso[24], a los píxeles de la ventana dependiendo de su cambio de textura y la profundidad a la que se encuentran.

Esta implementación utiliza SAD como métrica de similaridad (para ver la explicación de SAD y otros posibles sustitutos ver apartado Métricas de Stereo Matching), y permite regular el tamaño de la ventana a utilizar, pero obliga a que tenga forma cuadrada. Establece como tamaño mínimo 5 píxeles de lado.
La implementación de OpenCV permite modificar los valores de disparidad mínima y número de disparidades para permitir graduar el tamaño del volumen para el que se buscarán correspondencias. También permite modificar el tamaño de las ventanas de prefiltrado, el tipo de prefiltrado que se va a realizar, los valores límite de intensidad que se van a permitir.
Por último, se puede regular el paso de postprocesado estableciendo un valor de variabilidad de intensidad mínimo para los píxeles de una ventana, y el nivel de similaridad que se va a pedir para aceptar una correspondencia.

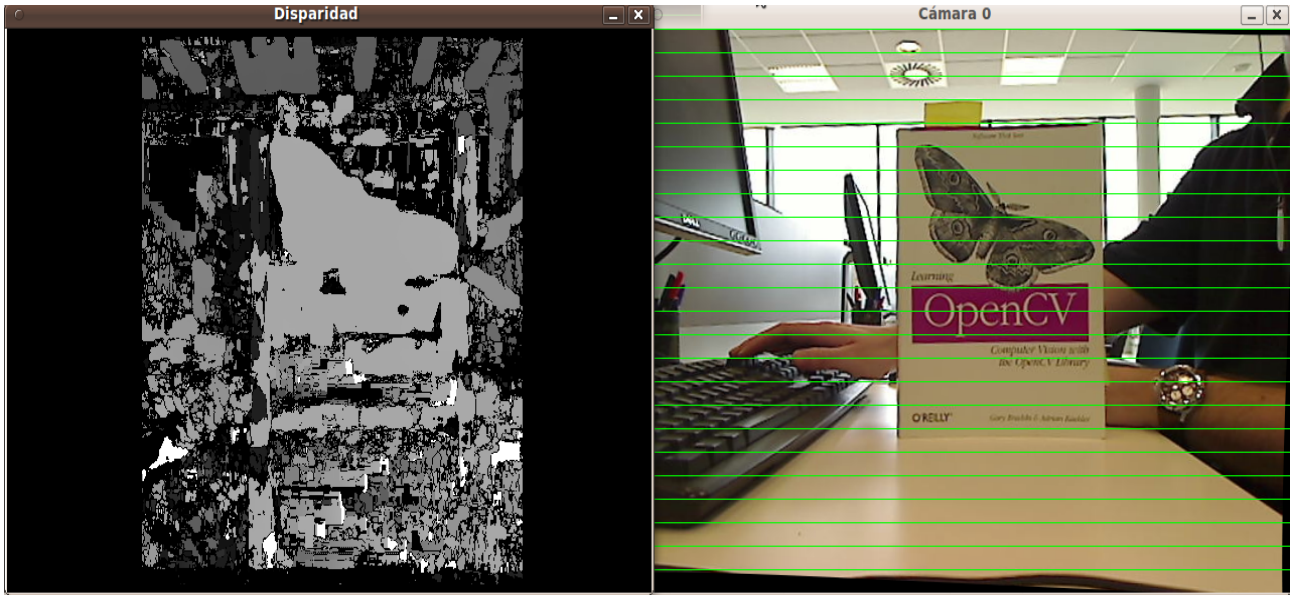

Ilustración 7: Mapa de disparidad calculado con Block matching

En la ilustración nº7 podemos ver un ejemplo de Block Matching. A la izquierda se presenta el mapa de disparidad generado por el algoritmo, y a la derecha una de las imágenes utilizadas para construir el mapa de disparidades.
Las zonas negras representan los puntos para los que no ha sido posible conseguir correspondencias, y los diversos grados de grises representan los píxeles para los que ha sido posible conseguirlas. La intensidad de gris marca el grado de disparidad de los bloques, siendo claro para los muy dispares, y oscuro para los que presentan una disparidad más pequeña.
Como se puede ver en la ilustración 7, las zonas con gran variedad de textura son fácilmente detectadas por el algoritmo (el caso de la mariposa), pero las superficies con intensidades uniformes son desechadas (mesa, ventanas...).

Correspondencia densa global, Graph-Cuts:
Los algoritmos de correspondencia globales intentan encontrar similaridades entre intensidades de píxeles a nivel de imagen completa, creando áreas de píxeles con el mismo nivel de similaridad. De esta forma, la correspondencia de un píxel afecta a sus vecinos.
Para ello intenta que dado un número de etiquetas, áreas de disparidad, etiquetar todos los píxeles de la imagen, de tal forma que la diferencia de las disparidades de los píxeles agrupados en una etiqueta tiendan a ser mínimas, y las diferencias entre disparidades de píxeles de distintas etiquetas sean lo mayores posibles, en un número infinito de iteraciones. En otras palabras, tratan de minimizar una función de energía a nivel de imagen completa.
Al agrupar píxeles en base a similaridad y profundidad, permite identificar formas y bordes con

gran precisión, evitando o reduciendo los problemas generados por oclusiones, cambios de iluminación, baja texturización... pero tiene la gran desventaja de que su cálculo es muy costoso, por lo que no puede llevarse a cabo en tiempo real.

Según la comparación de algoritmos llevada a cabo por la universidad de Middlebury en[5], los resultados más precisos son obtenidos por los obtenidos por métodos que utilizan Graph-cuts [25,26]. OpenCV cuenta con la implementación de un algoritmo de correspondencia global basado en Graph-cuts.

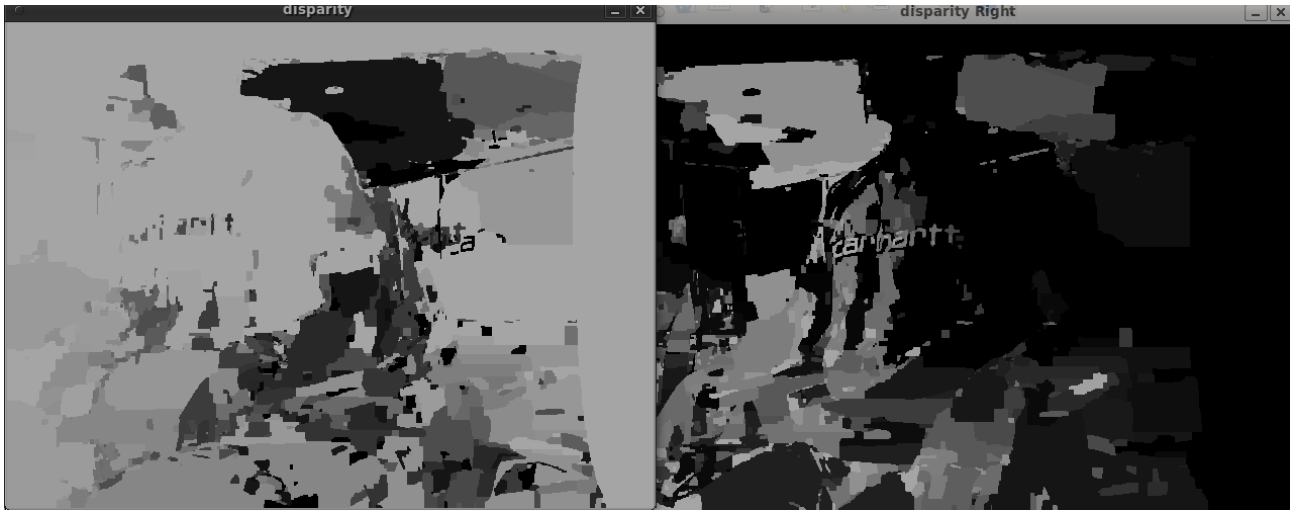

Ilustración 8: Ejemplo de resultado de cálculo de mapa de disparidades mediante Graph-cuts

En la implementación de Graph-cuts de OpenCV se pueden controlar los parámetros número de disparidades (con lo que se decide el número de áreas a asignar), y el número de iteraciones que el algoritmo va a disponer para refinar el resultado, cambiando píxeles de un área a otra para mejorar
Para el cálculo de este mapa, se han utilizado 256 disparidades, que se han calculado en 10 ciclos.
Para realizar el computo de un único *frame* (pareja de imágenes) han sido necesarios 20 minutos.
El tiempo de proceso necesario para el cálculo de un mapa de disparidades mediante GC es directamente proporcional al número de disparidades, y crece exponencialmente con el número de iteraciones.
La primera de las imágenes, *disparity*, muestra el mapa de profundidad en si, y la segunda imagen, etiquetada como *disparity Right*, muestra el mismo mapa de disparidad pero con valores invertidos.

Correspondencia densa híbrida, SGBM:
Este tercer tipo de algoritmos de correspondencia se caracteriza por ser un punto intermedio entre los algoritmos globales y los locales, intentando acercarse a la precisión de los globales sin por ello sacrificar la eficiencia computacional de los locales [27].
Para ello, dividen la imagen en ventanas, pero a la hora de calcular emparejamientos no buscan similaridades individuales entre píxeles, sino que intentan minimizar una función de energía a nivel de ventana.
Llamemos I_r imagen de referencia a una de las imágenes de una pareja estéreo rectificada, y I_t imagen objetivo (*target*) a la otra. Vamos a llamar p a la proyección de un punto en I_r y q a la proyección del mismo punto en I_t .
El punto p situado en el plano visual de la cámara de referencia tendrá coordenadas (x, y) y el punto $q=(x+d, y)$ donde d representa el desplazamiento horizontal.

Ahora vamos a tomar dos ventanas de n píxeles de lado, $w_n^r(i, j)$ y $w_n^t(i, j)$, en las imágenes I_r y I_t . También vamos a llamar a las parejas de ventanas:

$$w_n(i,j,d) = w_n^r(i,j), w_n^t(i+d,j)$$

Tomemos $S(p,q)$ como el conjunto de todos los píxeles que pueden ser evaluados a la hora de buscar correspondencias entre p y q. Cuando evaluemos correspondencias entre píxeles de ambas imágenes, estableceremos un subconjunto de píxeles en el cual buscaremos el píxel. Llamaremos a este subconjunto $S_v(p,q)$. Dentro de este subconjunto buscaremos la correspondencia minimizando una función de energía, al igual que en los algoritmos globales.

Algunos de estos algoritmos como el presentado en [28] permiten modificar el tamaño de la ventana n, para permitir correspondencias más precisas en zonas problemáticas, como bordes.

El algoritmo SGBM implementado en OpenCV es una modificación del HH08 presentado en[1] y que nos permite modificar una serie de parámetros para ajustar los resultados que obtendremos. Cuenta con los mismos parámetros que BM, pero a estos se le añaden dos parámetros de penalización, P1 y P2 que controlan la penalización por cambio de disparidad entre píxeles vecinos, y nos permiten regular la "suavidad" en los cambios de disparidad. P1 indica la penalización por el cambio de disparidad en 1 píxel, y P2 en más de un píxel. Por este motivo se recomienda que P1 sea menor que P2.
El algoritmo busca correspondencias mediante bloques, si bien permite reducir el tamaño de bloque hasta 1 píxel, en cuyo caso estaríamos buscando correspondencias mediante similaridad de píxeles. Los pasos de pre y postprocesado son los mismos que en el algoritmo BM.

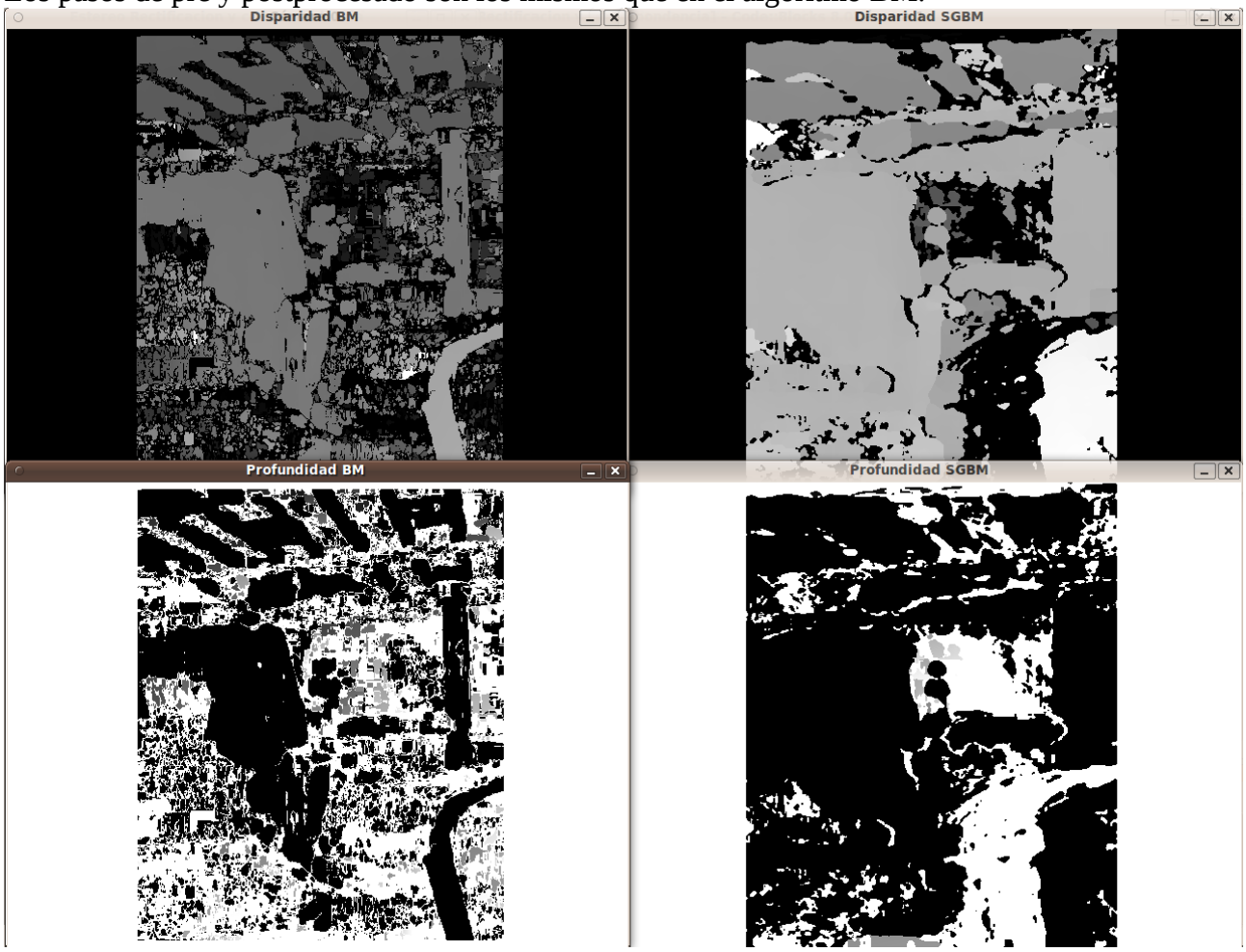

Ilustración 9: Comparativa de disparidades y profundidades generadas por BM y SGBM

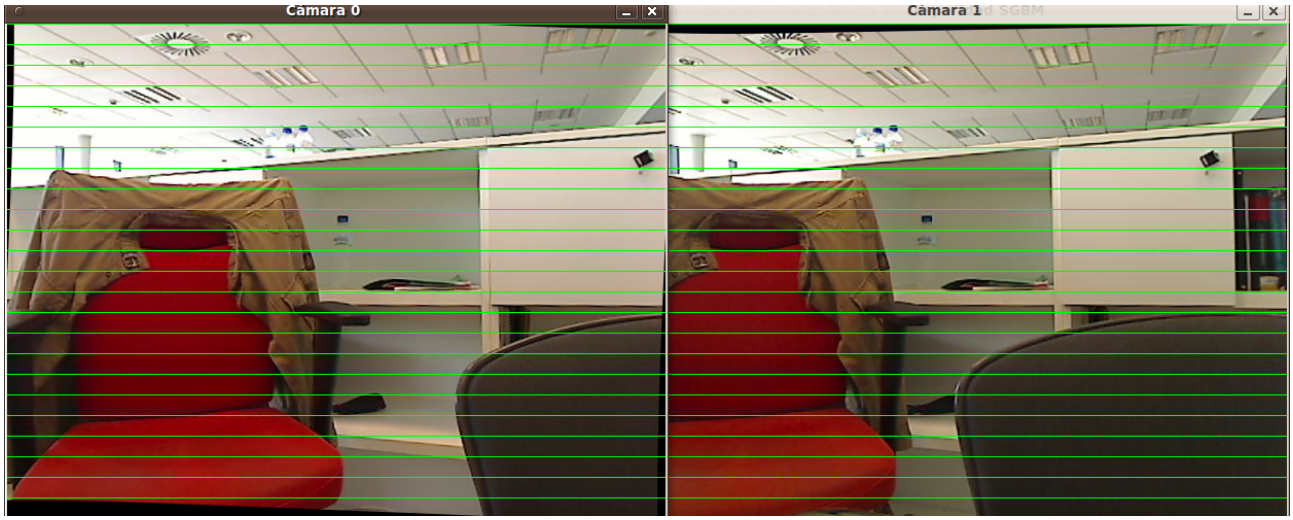

Ilustración 10: Imágenes utilizadas para calcular las disparidades

Como se puede ver en las ilustraciones 9 y 10, las diferencias en los resultados de las ejecuciones de BM y SGBM son notables. Donde existe una variación de textura relativamente grande (en la chaqueta, la etiqueta de la botella o en el borde de la silla, por ejemplo), BM no tiene ningún problema para encontrar las correspondencias, pero donde tenemos texturas más uniformes, la función de reducción de energía sí es capaz de detectar las correspondencias, y el SAD no genera una diferencia lo suficientemente significativa con los píxeles vecinos.

Rendimiento comparado de BM, GC y SGBM

Para comparar el rendimiento de los tres algoritmos de correspondencia soportados por OpenCV, y poder elegir cual utilizar en el proyecto, se ha implementado un programa que a partir de un par de imágenes comunes genera un mapa de disparidades mediante cada uno de los algoritmos. Con esto se ha podido comparar el grado de completitud para cada uno de los algoritmos.
Posteriormente, para poder medir el tiempo que cada algoritmo necesita para procesar un frame, se ha ejecutado una versión modificada de este programa de prueba, en la que solo se buscaba la disparidad mediante uno de los algoritmos.
Dado que carecemos del ground truth para las imágenes capturadas con nuestras cámaras, no ha sido posible comprobar la corrección de los mapas de disparidad.

Ilustración 11: Comparación de algoritmos de correspondencia

	Frames por segundo	Completitud
BM	4 FPS	40.63%
GC	485 segundos para procesar 1 frame	100%
SGBM	2 FPS	74.55%

Las imágenes utilizadas en este test tienen 640x480 píxeles.
A continuación se especifican los parámetros de ejecución para cada uno de los algoritmos:

Parámetros	Nº de disparidades	Nº de iteraciones
Graph-cuts	256	2

Parámetros	Tamaño ventana prefiltrado	Threshold prefiltrado	Tamaño ventana SAD	Valor mínimo de disparidad	Valor máximo de disparidad	Límite de variabilidad de textura	Ratio de unicidad de la textura	Uso de speckle
Block Matching	21	31	5	0	16	5	15	

Parámetros	Valor mínimo de disparidad	Valor máximo de disparidad	Tamaño ventana SAD	P1	P2	disp12Max Diff	Threshold prefiltrado	Ratio de unicidad de la textura	Uso de speckle	Full DP
SGBM	0	16	5	1800	7200	16	31	15	Desactivado	Desactivado

Se ha optado por dar un valor máximo de disparidad pequeño, para evitar posibles falsos positivos, así como para aumentar el área en la que se producen correspondencias.

Tras comparar el rendimiento de estos tres algoritmos se ha optado por utilizar SGBM en el proyecto, debido a que obtiene un grado de completitud muy alto sin penalizar excesivamente el tiempo de proceso.

Clasificación de algoritmos de correspondencia:

A la hora de clasificar el tipo de procesado de disparidad para los algoritmos que arrojan resultados discretos, se optado por clasificarlos como locales, ya que buscan correspondencias para píxeles o grupos de píxeles basándose únicamente en la similaridad de los píxeles vecinos.

Se ha clasificado el algoritmo Snakes como global debido a que su componente de energía de imagen tiene en cuenta las intensidades de toda la imagen. La métrica de similaridad compara la energía total de una serpiente con la energía total de otra.

Se ha incluido el algoritmo Harris por ser básico en el desarrollo de tracking en vídeo, y porque es el algoritmo que define qué es una esquina en una imagen[6], si bien su utilidad a la hora de detectar correspondencias entre imágenes estéreo es muy baja. A día de hoy es utilizado por algoritmos discretos como[29] para detección en la imagen de *corners*, de los que luego buscan la correspondencia en el resto de imágenes.

	Tipo de resultado	Tipo de procesado de disparidad	Métrica de similaridad
HH08: Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information	Denso	Híbrido	Función de energía
OpenCV: SGBM	Denso	Híbrido	Función de energía
OpenCV: BM	Denso	Local	SAD
OpenCV: Graph-cuts	Denso	Global	Función de energía
Snakes: active contour models	Denso	Global	SSD
SIFT: Distinctive Image Features from Scale-Invariant Keypoints	Discreto	Local	Distancia euclídea entre descriptores
SURF: Speeded Up Robust Features	Discreto	Local	Distancia euclídea entre descriptores
Shi Tomasi: Good Features to track	Discreto	Local	Distancia euclídea en espacio de imagen
Harris: A Combined Corner and Edge Detector	Discreto	Local	Distancia euclídea en espacio de imagen
Accurate, Dense, and Robust Multi-View Stereopsis	Discreto	Local	NCC
Adaptive Support-Weight Approach for Correspondence Search	Denso	Local	Weighted SAD
Multi-View Stereo Reconstruction and Scene Flow Estimation with a Global Image-Based Matching Score	Denso	Global	NCC y Mutual information
Stereo for Image-Based Rendering using Image Over-Segmentation	Denso	Global	
Fast Approximate Energy Minimization via Graph Cuts	Denso	Global	Función de energía (Graph-cuts)
Segment-based stereo matching using	Denso	Global	Función de energía

belief propagation and a self-adapting dissimilarity measure			(belief propagation)
Near real-time reliable stereo matching using programmable graphics hardware	Denso	Híbrido (programación dinámica)	

Propiedades de la correspondencia:

Unicidad:

Si pensamos en objetos opacos, un punto en una de las vistas debe tener como mucho una correspondencia en el otro. Puede darse el caso de que el punto esté fuera del cono de visión, u oculto detrás de algún otro objeto, por lo tanto solo hay 0 o 1 correspondencias entre puntos.

Continuidad:

Suponiendo que las superficies de los objetos suelen ser *smooth*, podemos indicar a la hora de afrontar el problema de reconstrucción para el punto P que $z = f(d)$ donde z es la distancia del punto P a las cámaras y d es la disparidad entre las cámaras.

Ordenamiento:

Tomemos las proyecciones de los puntos P, P' y P", que se encuentran en una recta, en el plano visual de la cámara 1. En el plano de imagen de la cámara 2 los puntos aparecerán en el mismo orden, y en el caso de que alguno de ellos no sea visible, por ejemplo debido a una oclusión, el resto seguirán apareciendo en el orden original.

Tomemos ahora los puntos M y N. M puede estar en cualquier punto de la zona de visión de las dos cámaras, pero situaremos N dentro del cono definido por las líneas Camara1-M y Camara2-M y dentro del mismo plano epipolar que M. El ordenamiento no se mantendrá. Se conoce a la zona donde sucede esto como la zona prohibida.

Siempre y cuando tengamos que aplicar esta regla a dos puntos que se encuentran dentro del mismo plano epipolar, tenemos que comprobar si serían visibles desde ambas cámaras en el caso de que se encontrasen en la superficie de un mismo objeto opaco.

Para más detalles ver [*Three Dimensional Computer Vision*] de Olivier Faugeras, pag. 178-179.

Gradiente de disparidad:

Llamamos así a la medida de la disparidad entre las vistas de dos cámaras. Se obtiene tomando dos puntos y aplicando la siguiente fórmula:

$$DG = \left| \frac{d_1 - d_2}{w_1 - w_2} \right|$$

Donde d representa la disparidad para dicho punto $d = v_2 - v_1$ y w es $\frac{v_1 + v_2}{2}$ .Los sistemas de visión tienen un gradiente de disparidad (DG) límite (conocido como K) a partir del cual muestran los puntos erróneamente, ya que la zona prohibida crece demasiado.

Constancia del color

Dado un punto del espacio físico y múltiples vistas del mismo, se supone que el color del mismo será igual para todas las vistas. Esta suposición, en la que se basan muchos métodos de correspondencia, por ejemplo los métodos en los que se comparan intensidades de píxeles entre las imágenes. Esta propiedad solo solo se da en superficies Lanbertianas, en las que no se forman brillos. Por tanto, en la practica totalidad de los materiales del mundo real no cumplen esta propiedad. Por este motivo Olivier Faugeras considera en [30] que es una suposición “inocente” que lleva a muchos errores de correlación.

Propiedades Geométricas de la Correspondencia:

Predicción de m_3 :

Tomemos que nuestro sistema tiene tres cámaras, y que el punto M es visible desde las tres. Conocida la proyección del punto en el plano de visión de la primera cámara (m_1) , sabemos que la proyección en la segunda estará dentro de una línea epipolar. Una vez hayamos localizado m_1 y m_2 , se generarán dos líneas epipolares en la tercera cámara. Si el punto es visible desde esta cámara, se encontrará el la intersección de las dos líneas, y por tanto no es necesario realizar una búsqueda para encontrarlo.

Mapas de Profundidad:

Una vez se ha calculado la correspondencia de los puntos de una imagen en la otra, o en las otras, queremos obtener la profundidad a la que se encuentran dichos puntos.

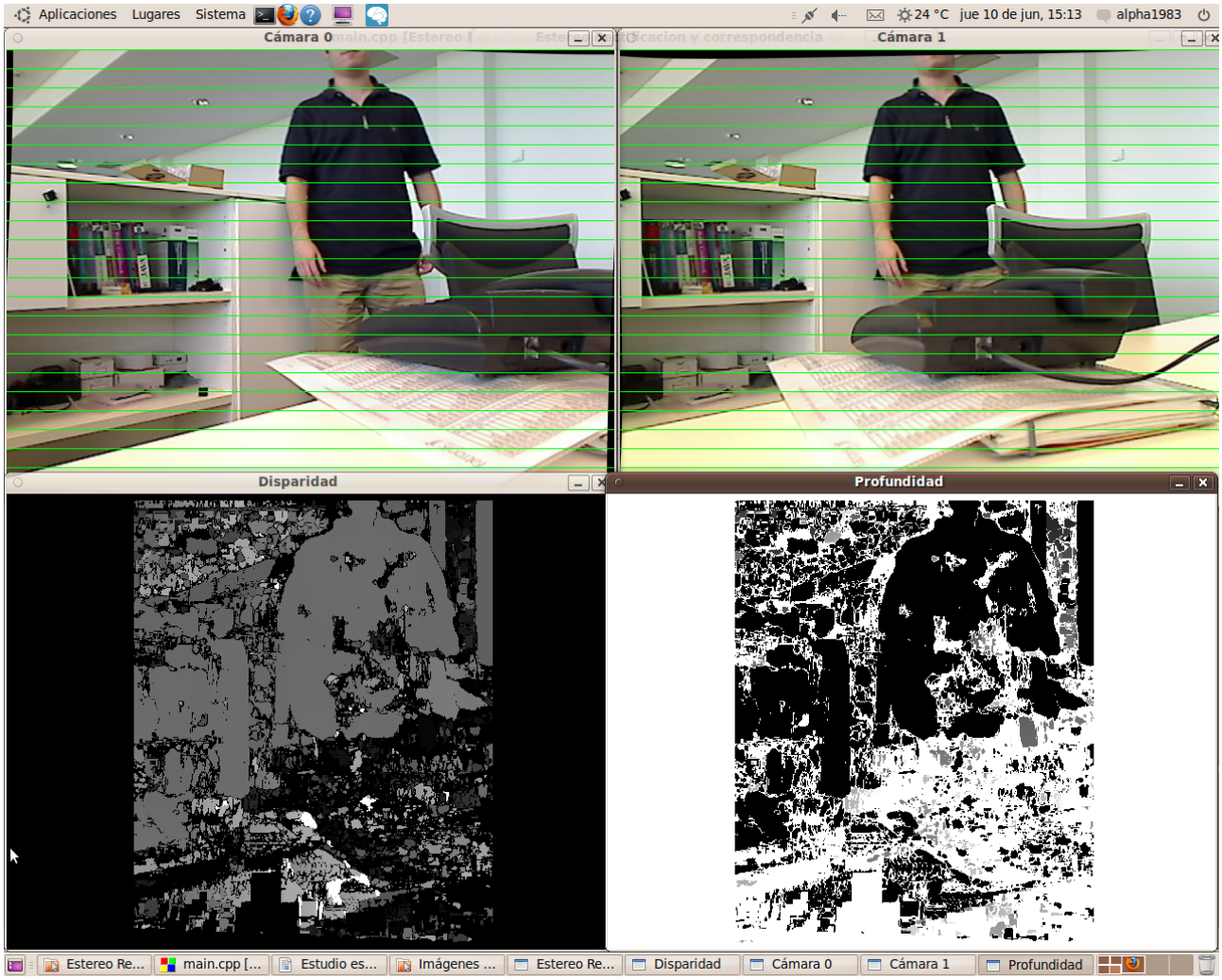

Ilustración 12: Mapa de profundidad

Tal y como se ve en la imagen, se calcula la profundidad de los puntos representados en el mapa de disparidades

Para el cálculo de la profundidad de un punto de la imagen son necesarios dos elementos, la matriz de reproyección Q (explicada al final del apartado dedicado al algoritmo de rectificación de Bouguet) y la disparidad para ese punto. Por tanto, solo podremos calcular la profundidad en aquellos puntos en los que hayamos identificado la correspondencia, y por tanto tengamos un valor de disparidad.

El cálculo de la profundidad se rige mediante la siguiente fórmula:

$$Q\begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T_x & (c_x - c'_x)/T_x \end{bmatrix} \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

Representaciones del entorno:

Problema a resolver

Hasta ahora hemos estudiado como recoger datos de un sistema estéreo y como procesarlos, pero entre la recogida y el procesamiento, y tras el procesamiento los datos han de ser almacenados en estructuras de datos. Dependiendo del uso que se vaya a dar a estos datos van a ser necesarias unas características en dichas estructuras de datos.

Como es lógico, nos interesa tener la estructura más simple posible para poder a continuación utilizar estos datos en el proceso para el que fueron recogidos.

Por tanto este problema tiene dos variables:

1. Determinar la información que debe ser representada.
2. Precisión con la que deben representarse los datos.

Es importante tener en cuenta que la precisión de los datos no solo cubre el número de decimales con los que se va a cuantificar el dato, sino también el grado de incertidumbre que tiene asociado dicho dato.

Para poder definir las dos variables, tenemos que responder a estas preguntas:

1. ¿Cuales son las primitivas del problema (volúmenes, superficies, puntos...)?
2. ¿Cual es el sistema de coordenadas de las primitivas? Para responder a esta pregunta dependemos de si el entorno a representar es rígido o dinámico.
3. ¿Como se estructuraran las primitivas en la estructura de datos? La efectividad de las búsquedas y operaciones que se realicen sobre los datos dependerán en gran medida de esto.
4. ¿Qué operaciones van a efectuarse sobre los datos? Algunas representaciones facilitan un tipo de operaciones y penalizan otras, por lo que el tipo de operaciones nos puede inclinar por una representación u otra.
5. Cual es el coste de construcción y almacenamiento de estas estructuras de datos?
    1. ¿Es necesaria toda la información de la imagen para representar toda la variabilidad que necesitamos?
    2. ¿Existe equivalencia entre la cuantía de un cambio en la representación y en el espacio físico? Dicho de otra forma, si a cambios físicos pequeños le corresponden cambios de representación grandes, el coste de computo y almacenamiento será grande.

Interpolación de superficies:

A la hora de crear superficies, podemos encontrarnos con que tenemos puntos problemáticos, en los que hay discontinuidades, o problemas de orientación. Estos puntos problemáticos también pueden deberse a que son visibles desde pocas vistas, o que la texturización del punto es demasiado pobre. Para solucionar estos problemas se interpolan los valores de la superficie, esto es, en base a los valores de puntos cercanos conocidos estimamos el valor de los puntos desconocidos.

Tipos de representación de entorno: Vóxeles

Llamamos voxel (*volumetric pixel*) a una estructura tridimensional que representa volumen. Mediante la combinación de múltiples vóxeles es posible crear objetos o escenas tridimensionales.

A la hora de representar un entorno, nos permiten representar su superficie, y su contenido, por lo que son muy utilizados en campos como la imagen médica, en los que es preciso ver las diferentes capas de los órganos.

Suele ser el sistema de representación elegido por los algoritmos de reconstrucción “escultores”, que parten de un volumen que contiene la pieza o la escena a representar, y van esculpiéndolo para acercarse al volumen real.

Los vóxeles limitarán el nivel de precisión de la representación, ya que la máxima precisión que podemos mostrar está limitada por el tamaño del voxel, pero la representación de la escena siempre será densa.

Debido a que los vóxeles deben cubrir el volumen completo del objeto, es necesario procesar mucha más información que en el caso de que solo estemos tratando la superficie de los objetos. Por otra parte, debido a que cubre la totalidad del volumen del objeto, son muy utilizados cuando es necesario representar el interior de un objeto, por ejemplo en campos como la imagen médica.

En la clasificación de algoritmos de reconstrucción hemos englobado los vóxeles dentro de la categoría de volúmenes.

Tipos de representación de entorno: Level sets

Representación en la que partimos la escena en niveles de profundidad o planos, y registramos todos los puntos en los que los objetos intersectan con los planos. Por tanto estamos ante una representación densa del entorno.

La precisión con la que podemos representar la escena estará limitada por la distancia física entre niveles de profundidad.

En la clasificación de algoritmos de reconstrucción hemos englobado los *level sets* dentro de la categoría de volúmenes.

Tipos de representación de entorno: Mapas de profundidad

Un mapa de profundidad es una matriz en la que registramos las distancias desde el sistema de cámaras hasta los puntos del objeto para los que existen mediciones.

Permiten representaciones tanto densas como discretas. Consideraremos que un mapa de profundidad es denso cuando se representen profundidades individuales para cada píxel de las cámaras del sistema, y discreto en caso contrario.

Permite representar únicamente la superficie del objeto o la escena, y suelen ser utilizados por algoritmos iterativos.

Una de sus grandes desventajas, es que los puntos son independientes entre si, por lo que no obliga a que se tengan en cuenta los valores de los puntos cercanos, lo cual puede llevar a la aparición de discontinuidades.

Normalmente suele utilizarse durante el cálculo de la reconstrucción, y luego se convierte en una malla poligonal para poder representar gráficamente los resultados.

Tipos de representación de entorno: Malla poligonal deformable

Consiste en un conjunto de puntos en formato matricial para los que se conoce su posición, pero utilizando las posiciones de estos puntos, se establecen planos (3 puntos por cada plano). De esta forma, partiendo de información discreta (posiciones de puntos) podemos representar superficies continuas.

Es utilizado tanto durante el proceso de cálculo de la reconstrucción, como durante la representación de la reconstrucción.

Técnicas de Visión Estéreo:

Reconstrucción: funciones de energía

Nombre que proviene del campo de las matemáticas. Representa una función que buscamos minimizar para optimizar un proceso. En este caso, la se busca minimizar la diferencia entre el modelo reconstruido, y el objeto o escena en el espacio físico.

Reconstrucción: Función de coste de superficie

Algoritmo de reconstrucción iterativo que trata de obtener la superficie del objeto modificando la estimación de la superficie tratando de reducir la disparidad de los puntos de la superficie con lo observado por las cámaras. Suelen utilizar mapas de profundidad o mallas poligonales para representar las superficies.

Reconstrucción: Función de coste de volumen

Algoritmo de reconstrucción que calcula la modificación que a aplicar al volumen completo para reducir una función de coste en una iteración hasta alcanzar un determinado límite (y de varias pasadas alcanzar la superficie real del objeto). Otras variantes intentan llegar a la superficie real del objeto en una única iteración, y para ello tratan de hallar el mínimo global de la función de energía.

Reconstrucción: Escenas

Decimos que una reconstrucción es capaz de reconstruir escenas cuando es capaz de reconstruir el entorno partiendo de un conjunto de vistas tomadas desde unos ángulos limitados. Las escenas suelen estar formadas por múltiples objetos

Reconstrucción: Objetos

Decimos que una reconstrucción reconstruye únicamente objetos cuando el algoritmo necesita vistas de todo el entorno del objeto para poder generar su modelo.

Reconstrucción discreta: Estimación de pose

El objetivo de estas reconstrucciones es estimar la pose de una serie de objetos en la escena. Para ello reconstruyen puntos de la escena, y a continuación buscan el objeto al que corresponden. Una vez identificados un conjunto de puntos como pertenecientes a un objeto, se calcula la posición de estos puntos en el propio objeto. Tras esto obtenemos la posición en el espacio físico del objeto en si y la orientación (*pitch, roll & yaw*) del mismo.

Estos algoritmos de reconstrucción son utilizados en robótica para permitir que brazos robóticos puedan reconocer y coger objetos. Un ejemplo de estas aplicaciones sería [31].

Reconstrucción discreta: Densificación

Una vez se ha generado una reconstrucción discreta del objeto o de la escena, y se tienen un conjunto de puntos de la superficie del objeto o de la escena, es posible interpolar la posición del resto de puntos para obtener una superficie densa.

En las zonas vacías que se encuentran ente los puntos reconstruidos, es posible “expandir” los puntos conocidos más cercanos para obtener una aproximación a la superficie real. Un ejemplo de este tipo de técnicas sería el algoritmo presentado por Furukawa en [29].

Clasificación de Algoritmos de Reconstrucción

	Tipo Resultados	Necesidades previas del algoritmo	Tipo de Algoritmo de Reconstrucción	Algoritmo Estimación Fotométrica	Sistema de Representación	Gestión de Oclusiones
Accurate, Dense, and Robust Multi-View Stereopsis	Densificado Objetos	Visual-hull	Densificación		Patches y Malla Poligonal	
Multi-View Stereo Reconstruction and Scene Flow Estimation with a Global Image-Based Matching Score	Denso	Eliminar oclusiones	Función de energía (gradient descent)	Espacio de imagen	Volumen (level sets)	Ninguna, se eliminan antes de llegar al algoritmo
Silhouette and Stereo Fusion for 3D Object Modeling	Denso Objetos	Silhouette	Función de coste de superficie (iterativa)	Espacio de escena	Volumen	Outlier
Multi-view Stereo via Volumetric Graph-cuts	Denso Escena	Visual-hull o estimación de superficie	Función de coste de volumen	Espacio de escena	Volumen (voxels)	Geométrica Calculado a partir del visual-hull
Multi-View Stereo Revisited	Denso Objeto	Bounding-box o visual-hull	Fusión de mapas de profundidad	Espacio de escena	Mapas de profundidad	Outlier
Combined Depth and Outlier Estimation in Multi-View Stereo	Denso Escena	Parámetros intrínsecos y extrinsecos de las cámaras		Espacio de imagen		Geométrica Calculado durante el proceso de correspondencia
Accurate Dense Stereo	Denso Escena	Ninguna		Espacio de escena	Malla poligonal	

Reconstruction using Graphics Hardware						
Robust high precision 6D pose determination in complex environments for robotic manipulation	Discreto Escena	Ninguna	Estimación de pose	Espacio de escena	Features (SIFT)	Outlier
Manipulator and Object Tracking for In Hand Model Acquisition	Discreto Escena	Ninguna	Estimación de pose		Features (SIFT)	Outlier

Structure From Motion:

Hasta ahora hemos trabajado con sistemas en los que conocíamos la posición de las cámaras, y a partir de las mismas calculábamos las posiciones de los puntos visualizados. Pero en vez de eso nos podemos encontrar que desconocemos la posición de la cámara en los momentos en los que obtuvo las imágenes que estamos analizando. Para poder localizar las posiciones de las cámaras tomamos una serie de puntos en las imágenes, y buscamos correspondencias entre ellos, y a continuación hallamos la disparidad para los puntos. Con estos datos calculamos la posición de la cámara, la reconstrucción de los puntos y los parámetros de calibración de la cámara.

Este sistema puede utilizarse tanto en sistemas monoculares como multicámara.

En nuestro proyecto vamos a contar con un sistema estéreo en movimiento, donde conoceremos la posición relativa entre las dos cámaras, pero no la posición del sistema de visión, que deberá ser calculado.

Este algoritmo nos pide que le suministremos un conjunto de imágenes para poder buscar los parámetros, pero para nuestro ejemplo supondremos que tenemos tan solo dos, en las cuales o bien se ha movido la cámara, o el objeto/escena observado o ambos se han movido.

Para cada una de las imágenes seleccionamos *features*, y buscamos correspondencias entre las *features*. Una vez tenemos correspondencias, tenemos que:

$$P = p_l M_l^{-1} = p_r E M_r^{-1}$$

Esto es, para unas coordenadas de plano de imagen conocidas p_l y p_r que sabemos por la correspondencia que son las proyecciones en de un mismo punto del espacio físico P queremos calcular la matriz de cámara y el cambio de posición de cámara ente una imagen y otra, y de paso obtendremos también las coordenadas físicas de P .

Para poder hacer este cálculo, vamos a buscar correspondencias de un gran número de puntos, y vamos a buscar los parámetros que hagan que se generen proyecciones válidas para todos estos puntos. Además, queremos que exista desplazamiento entre los puntos detectados entre una imagen y otra, ya que de lo contrario no tendremos utilizar el punto para efectuar ningún cálculo. Por este motivo decimos que obtenemos la estructura a partir del movimiento, ya que es mediante el cambio de posición de los puntos entre imágenes que conseguimos la información necesaria para estimar la posición de las cámaras y de los puntos.

Este proceso se enfoca como una función de búsqueda, en al que queremos llegar a un mínimo global (en principio de valor 0, pero no alcanzaremos ese valor en la realidad) de error de reproyección (error en el que medimos en que grado las proyecciones en el mundo físico de los puntos observados en las imágenes no coinciden). El espacio de búsqueda está compuesto por los parámetros que buscamos y la posición física del punto proyectado. Además, la función de error suele tener muchos mínimos locales, por lo que es fácil que la función de búsqueda se encamine a un mínimo local.

El tiempo necesario para encontrar el valor mínimo es directamente proporcional al número de parámetros que tiene que calcular, por lo que el conocimiento previo de parte de estos parámetros ayuda a reducir el tiempo necesario para realizar el cálculo. Por ejemplo, si sabemos que todas las imágenes han sido tomadas por una cámara, o cuales de las imágenes han sido tomadas por que cámara, podemos tomar valores comunes para la matriz de cámara M .

A la hora de realizar la calibración de las cámaras del sistema estéreo utilizábamos imágenes en las que podía verse un tablero de ajedrez, y tomábamos como *features* los puntos de unión entre las celdas.

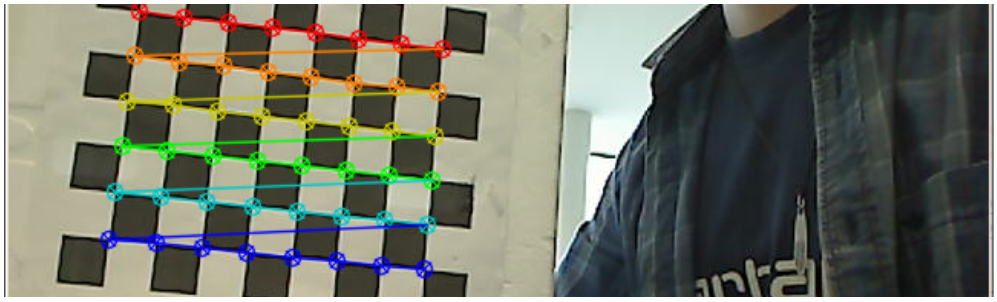

Ilustración 13: Features utilizadas en structure from motion

Dado que sabemos que los puntos físicos P cuyas proyecciones estamos utilizando están en un mismo plano físico, podemos eliminar todas las combinaciones de parámetros en las que los puntos proyectados a espacio físico no están en un plano común, lo cual simplifica notablemente la búsqueda del resto de los valores.

Las funciones de calibración presentes en OpenCV utilizan este sistema para obtener los parámetros matriz de cámara (distancia focal y centro óptico) y parámetros de distorsión y la función de calibración estéreo además calcula matriz de rotación entre las dos cámaras y vector de traslación entre las cámaras.

La función de calibración estéreo de OpenCV introduce una restricción adicional para facilitar la obtención de los parámetros. Solo permite que las cámaras se encuentren en un mismo plano, y además que se encuentren en un eje horizontal o vertical. De esta forma tan solo se tiene que calcular una de las tres componentes del vector de traslación T , la x o la y, dependiendo de si tenemos estéreo verticales u horizontal.

Por último hay que especificar que los resultados obtenidos por *structure from motion* siempre tendrán un cierto grado de error, debido a que a lo largo del proceso hay muchos puntos en los que se va perdiendo precisión y acumulando grados de error [13]. Por ejemplo, estamos situando la proyección de P en el plano de imagen exactamente en un píxel, cuando puede darse en el espacio sub-píxel, o podemos tener errores de correspondencia que generen modelos no válidos e introduzcan error en la estimación. Este problema es afrontado por los algoritmos de Bundle Adjustment.

Bundle Adjustment

Se llama así al problema de refinar la reconstrucción visual para producir una estructura tridimensional y unos parámetros de visualización (pose y calibración de cámara) óptimos [12]. Esto es, se intenta que el error conjunto de la reconstrucción y de los parámetros de visualización sea el mínimo posible.

Para conseguir reducir el error de reproyección se utiliza un algoritmo de minimización no lineal robusto que modifica los valores estimados de los parámetros conjuntamente [4], o dicho de otra forma, intenta ajustar el grupo (*bundle*) de rayos que unen las proyecciones y los puntos físicos de tal forma que el error de reproyección sea el menor posible. Utilizando una técnica que tenga en cuenta estas relaciones entre parámetros se consiguen mejoras en flexibilidad, precisión y eficiencia frente a algoritmos normales de *structure from motion* [12]. Los algoritmos de *structure from motion* acumulan error frente a los de Bundle Adjustment, que intentan reducirlo al mínimo.

Además de obtener mejores resultados, BA está más optimizado de cara a las necesidades de procesamiento. Por ejemplo, una de las mejoras que permite este sistema es que si tenemos una o varias cámaras en movimiento, sabemos que una determinada *feature* solo será visible durante un tiempo y luego desaparecerá, por lo que si no hemos sido capaces de detectar la correspondencia de dicha *feature* desde hace k *frames*, podemos ignorarla.

Algoritmo de Levenberg-Marquat

Entre los algoritmos de Bundle Adjustment el más conocido [13] es el algoritmo de Levenberg-Marquat (LM). Es un algoritmo iterativo que minimiza una función de coste no lineal, en nuestro caso el error de reproyección, modificando los valores de un conjunto de parámetros. Para poder empezar con la minimización, LM necesita un punto de partida, un conjunto de valores para los parámetros y las mediciones de error para dichos parámetros. A cada iteración de LM se calcula un nuevo conjunto de parámetros que reduce el error, hasta llegar a la solución óptima, y la modificación que se produce entre una iteración y la siguiente es aquella que minimiza el error en un cierto grado. Ese límite de reducción de error (LRE) nos sirve para evitar que el sistema coja cada vez el mejor valor y acabe realizando una búsqueda *greedy.* La particularidad de este algoritmo consiste en que el LRE se recalcula en cada iteración, de tal forma que si en una iteración se encuentra una modificación de parámetros que reduzca el error, en la siguiente LRE tendrá un valor menor, y si en una iteración no encuentra una modificación que satisfaga a LRE, lo aumenta.

De esta forma, gracias al uso del límite de reducción de error LM presenta un comportamiento que oscila entre el de una búsqueda *greedy* y una *Gauss-Newton least squares update* [13].

Mejora de estimaciones de posición

Structure from motion también nos permite refinar los resultados de la triangulación de un punto visto en diferentes imágenes. Tal y como se explicó en el apartado de triangulación, la precisión de la estimación de las coordenadas espaciales de un punto están limitadas por el tamaño del baseline y la precisión de medición de los ángulos, por lo que para aumentar la precisión en el cálculo de las coordenadas tenemos que aumentar el tamaño del baseline.

Pongamos que tenemos dos cámaras e ignoramos la posición relativa entre ambas, esto es, la matriz R y el vector t que nos permiten identificar las correspondencias de los píxeles de un plano de imagen y otro. Esta información también está contenida en la matriz esencial (E).

Digamos que tenemos N observaciones en las que hemos identificado un punto en ambas imágenes, y sabemos que los elementos que transforman los píxeles de una imagen en los píxeles de la otra son R y t, o la matriz E que los engloba a ambos. Por tanto, si tenemos suficientes observaciones podremos crear un sistema de ecuaciones que nos permita resolver el valor de los elementos de la matriz E.

$$x_{i0}\,x_{i1}\,e_{00}+y_{i0}\,x_{i1}\,e_{01}+x_{i1}\,e_{02}+x_{i0}\,y_{i1}\,e_{10}+y_{i0}\,y_{i1}\,e_{11}+y_{i1}\,e_{12}+x_{i0}\,e_{20}+y_{i0}\,e_{21}+e_{22}=0$$

Donde x_{i0} representa la coordenada x de la i-esima observación (de un total de N) en la imagen 0, y e_{00} el valor de las coordenadas (0,0) de la matriz E.

Como la anterior ecuación presenta 9 incógnitas (todos los elementos de la matriz E), necesitaremos 9 ecuaciones como esta, por lo que el número mínimo de correspondencias distintas entre el conjunto de imágenes de las dos cámaras para hallar la matriz E es 9.

De esta forma podemos obtener los parámetros necesarios para generar un nuevo sistema estéreo en el cual tomamos como planos visuales de las cámaras imágenes situadas a distancias mayores que las presentes en el baseline de las cámaras reales.

Técnicas de Postprocesado:

Los algoritmos aquí presentados son utilizados para mejorar o filtrar los resultados obtenidos de reconstrucciones 3D mediante visión mono o estéreo.

Range data merging:

Consiste en juntar las diferentes profundidades obtenidas para poder generar superficies. Puede hacerse para unir dos reconstrucciones generadas mediante conjuntos de datos distintos y con ello generar una superficie mayor, o para mejorar la fiabilidad de las medidas al compararlas con otras.

La técnica más utilizada es *Iterated Closest Point* (ICP), que permite encontrar los puntos de unión de dos superficies, y orientarlas para generar una única superficie. Con esto se consigue alinear las dos superficies, unirlas y generar una superficie que cubra un área mayor del objeto o la escena a representar.

Es común que surjan problemas debido a la presencia de malas medidas, especialmente en los bordes de las superficies a unir. En esos casos, se deben detectar estas mediciones corruptas y eliminarlas del modelo final. Para ello se suele hacer uso de técnicas de tratamiento de outlayers como las presentadas en [32].

Diccionario:

Fotoconsistencia:

Mide lo correcta que es una reconstrucción en un punto o en su totalidad. Existen diferentes métodos para medirla, pero normalmente suele realizarse comparando imágenes al modelo generado.

Sum of Absolute Differences:

Es una métrica utilizada por los algoritmos de correspondencia para medir similaridades. En este algoritmo se genera una ventana alrededor del píxel que queremos detectar en la otra imagen, y a continuación se calcula la suma de las intensidades de todos los píxeles de esa ventana. En la otra imagen, para cada píxel de esa ventana que se encuentre en la línea epipolar, se genera una ventana del mismo tamaño que la del píxel que queremos detectar, y entonces se suman los valores de las intensidades de todos los píxeles de la ventana. A continuación calculamos el valor absoluto de la diferencia entre la suma de intensidades de la ventana del píxel original y la de cada uno de los píxeles de la línea epipolar.

También se conoce a este algoritmo por sus siglas, SAD.

Sum of Squared Differences:

Es una métrica utilizada por los algoritmos de correspondencia para medir similaridades. El algoritmo funciona de forma similar a SAD, salvo que en vez de buscar el valor absoluto, elevamos al cuadrado la diferencia de ventanas. Es mucho más lento que el cálculo de SAD, debido a que hay que hacer una multiplicación adicional por cada pareja de píxeles que comparamos.

También se lo conoce como SSD.

Conclusiones:

En este proyecto se ha abordado el problema de obtener información tridimensional a partir de información bidimiensional. Para ello se han investigado problemas como correspondencia discreta, correspondencia estéreo, correspondencia multiview, reconstrucción discreta y densa, así como el cálculo de desplazamiento y rotación del sistema de cámaras estéreo entre frames.

Para la correspondencia discreta, se han investigado los algoritmos SIFT y SURF, habiéndose elegido el primero para su implementación. En correpondencia densa, se han evaluado los algoritmos BM, SGBM y Graph-cuts, habiéndose seleccionado SGBM por su relación de porcentaje de completitud y tiempo de procesado.

Las aplicaciones de este proyecto son muy variadas, desde el modelado automático de objetos reales para su uso en sistemas virtuales a la robótica. Los resultados obtenidos en este proyecto indican que es posible realizar reconstrucciones densas del entorno en tiempo real o en casi tiempo real, utilizando equipos de escritorio. Estas conclusiones se han visto también avaladas por [17], que utilizando un método similar al estudiado en este proyecto, si bien mucho más optimizado, es capaz de realizar reconstrucciones densas del entorno en tiempo real utilizando múltiples imágenes obtenidas con una única cámara y relacionadas entre si mediante bundle adjustment.

Con este tipo de reconstrucciones se podrían obtener mapas del entorno mucho más fiables que los obtenidos con métodos de reconstrucción discreta como SLAM. De esta forma, la percepción del entorno no estaría limitada a las *features* salientes, y podría pasar a dependerse únicamente de los sistemas de visión para la evasión de obstáculos, evitando de esta forma los gastos que suponen los sistemas de medición de tiempo de vuelo, como puede ser el láser.

Estos métodos también tendrían otra ventaja sobre el láser, debido a que un sensor de este tipo solo va a obtener mediciones del entorno para una serie de arcos concretos, por lo que si el obstáculo no se encuentra en la trayectoria de dicho haz, no será percibido. Por el contrario, con un sistema de cámaras, el grado de entorno percibido por el sistema es mucho mayor, por lo que es mucho más difícil que un obstáculo no sea percibido.

Bibliografía:

[1] Heiko Hirschmüller, “Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information.”

[2] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, y Richard Szeliski, “A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms.”

[3] Daniel Scharstein y Richard Szeliski, “A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms.”

[4] Richard Szeliski, *Computer Vision: Algorithms and Applications*.

[5] Evaluación de algoritmos de reconstrucción de la universidad de Middlebury http://vision.middlebury.edu/mview/eval/

[6] Gary Bradski, Adrian Kaehler, *Learning OpenCV*.

[7] Federico Tombari, “Methodologies for visual correspondence.”

[8] D. Geiger, B. Ladendorf, y A. Yuille, “Occlusions and binocular stereo,” *International Journal of Computer Vision*, vol. 14, Abr. 1995, págs. 211-226.

[9] Michael Kass, Andrew Witkin, y Demetri Terzopoulos, “Snakes, Active Contour Models,” 1987.

[10] Richard Hartley, “Theory and Practice of Projective Rectification.”

[11] Olivier Faugeras, *Three-Dimensional Computer Vision*.

[12] B. Triggs, P. Mclauchlan, R. Hartley, y A. Fitzgibbon, “Bundle Adjustment -- A Modern Synthesis,” 2000.

[13] T. Moons, “3D Reconstruction from Multiple Images” *Foundations and Trends® in Computer Graphics and Vision*, vol. 4, 2008, págs. 287-404.

[14] H.C. Longuet-Higgins, “A computer algorithm for reconstructing a scene from two projections,” *Nature*, vol. 293, 1981, págs. 133-135.

[15] F. Dornaika, “CONTRIBUTIONS A L'INTEGRATION VISION ROBOTIQUE : CALIBRAGE, LOCALISATION ET ASSERVISSEMENT ”, presentada en *INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE* el 25 de septiembre de 1995

[16] Richard O. Duda, Peter E. Hart y David G. Stork, *Pattern Classification*, segunda edición, 2001, de Willey

[17] Richard A. Newcombe y Andrew J. Davison, “*Live Dense Reconstruction with a Single Moving Camera*”,2010

[18] E. Murphy-Chutorian, M. Manubhai “*Head Pose Estimation: A survey*” *Pattern Analysis and Machine Intelligence*, vol 31 n 4, Abr. 2009, págs. 607-626.

[19] C. Harris y M. Stephens, “*A Combined Corner and Edge Detector” In. Alvey Vision Conference*, páginas 147–152, 1988

[20] A. Lowe, “*Distinctive Image Features from Scale-Invariant Keypoints ” International Journal of Computer Vision*, 60, 2 (2004)

[21] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool, *"SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU)*, Vol. 110, No. 3, pp. 346--359, 2008

[22] Furukawa, Y. and Ponce, J. 2009. Accurate Camera Calibration from Multi-View Stereo and Bundle Adjustment. *Int. J. Comput. Vision* 84, 3 (Sep. 2009)

[23] *T. Kanade, M. Okutomi “A Stereo Matching Algorithm with an Adaptive Window: Theory and Experiment” Proceedings ICRA*, April, 1991, pp. 1088-1095.

[24] K. Yoon and I. S. Kweon, *"Adaptive Support-Weight Approach for Correspondence Search," IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 28, no. 4, pp.

650-656, 2006

[25] G. Vogiatzis, C. Hernández, P. H. S. Torr, and R. Cipolla. "*Multi-view Stereo via Volumetric Graph-cuts" IEEE Transactions in Pattern Analysis and Machine Intelligence* ***(PAMI)***, vol. 29, no. 12, pages 2241-2246, Dec., 2007.

[26] S. Tran, L. Davis "3D surface reconstruction using graph cuts with surface constraints" EECV pages 219-231, 2009

[27] F. Tombari "PhD Thesis: Methodologies for visual correspondence", presentada en la UNIVERSIT A DI BOLOGNA en diciembre de 2008

[28] D. Geiger, B. Ladendorf, A. Yuille *"Occlusions and binocular stereo" International Journal of Computer Vision* Volume 14 , Issue 3 , Pages: 211 - 226 (April 1995)

[29] Y. Furukawa, J. Ponce *Accurate, Dense, and Robust Multi-View Stereopsis Computer Vision and Pattern Recognition, 2007. CVPR '07.* pp. 1-8.

[30] R. Keriven, O. Faugeras, *"Multi-View Stereo Reconstruction and Scene Flow Estimation with a Global Image-Based Matching Score"*, The International Journal of Computer Vision, vol 72, 2006

[31] T. Grundmann, R. Eidenberger, M. Schneider, M. Fiegert, G. v. Wichert *"Robust high precision 6D pose determination in complex environments for robotic manipulation"*, ICRA 2010

[32] M. Goesele, B. Curless, S. M. Seitz "Multi-view Stereo Revisited" Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 Pages: 2402 – 2409, 2006

[33] M. I. A. Lourakis, A. A. Argyros, *"SBA: A Software Package for Generic Sparse Bundle Adjustement"*, ACM Trans. Math. Software, vol 36, nº 1, pag 1-30, 2009