Trabajo de Fin de Grado

Sistema de odometría visual: Estimación de la localización de un vehículo.

Visual odometry system: Estimation of vehicle´s location.

Autor: Noel Díaz Mesa

La Laguna, 8 de septiembre de 2015

D. **Jesús Miguel Torres Jorge**, con N.I.F.: 43.826.207-Y profesor contratado como Doctor y adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **José Demetrio Piñeiro Vera**, con N.I.F.: 43.774.048-B profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor.

CERTIFICAN

Que la presente memoria titulada:
"Sistema de odometría visual: Estimación de la localización de un vehículo."
ha sido realizada bajo su dirección por D. **Noel Díaz Mesa**, con N.I.F. 78.619.072-N.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de septiembre de 2015

Agradecimientos

A mi tutor por la ayuda en el desarrollo de este trabajo final de grado, a mi familia por haberme apoyado durante el desarrollo de mis estudios universitarios, en especial a mi mujer y mis dos preciosas hijas que son las que me dan la fuerza para enfrentarme a cualquier obstáculo.

Resumen

El objetivo del presente proyecto es el desarrollo e implementación de un sistema de localización para vehículos o robots móviles basado en odometría visual, consistente en el seguimiento de puntos característicos en las imágenes obtenidas a través de un sistema de visión monocular. Es decir, aquellas que usan las imágenes de una sola cámara. Implementado exclusivamente con la librería OpenCV y el lenguaje de programación Python.

Abstract

The objective of this project is the development and implementation of a tracking system for vehicles or mobile robots based on visual odometry, consisting of tracking feature points in the images obtained through a monocular vision system. That is, those that use images from a single camera. Implemented exclusively with the library OpenCV and Python programming language.

Licencia

Índice General

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Introducción

En navegación, la odometría es el uso de los datos de movimiento de los actuadores, medidos a través de dispositivos tales como encoders giratorios, para estimar el cambio en la posición a lo largo del tiempo. Si bien esto es útil para muchos vehículos con ruedas, las técnicas de odometría tradicionales no se pueden aplicar a los robots móviles con métodos de locomoción no estándar, tales como aquellos que usan patas.

Además la odometría sufre universalmente de problemas de precisión, puesto que las ruedas tienden a deslizarse sobre el suelo, dando una distancia de viaje no uniforme en comparación con la que se obtiene a partir de la medida de la rotación las ruedas. Este error es aún mayor cuando el vehículo opera en superficies no lisas ya que las lecturas de odometría se vuelven cada vez menos fiables con el tiempo, dado que los errores se acumulan a lo largo del tiempo.

En la robótica y visión por computador, la odometría visual es el proceso de determinar la posición y orientación de un robot mediante el análisis de las imágenes de las cámaras incorporadas. Siendo una técnica que se ha utilizado en una amplia variedad de aplicaciones de robótica, como en los Mars Exploration Rovers.

Figura 1: La Mars Science Laboratory.

La odometría visual es el proceso de obtener información de odometría, como hemos comentado, utilizando una secuencia de imágenes. Esta técnica permite mejorar la precisión de la navegación independientemente del tipo de locomoción y de la superficie.

1.2 Estado del arte.

En la actualidad son muchos los trabajos desarrollados sobre sistemas de odometría visual, desde sistemas estereoscópicos o monoculares a sistemas SLAM, del Inglés Simultaneous Localization And Mapping. En este apartado se expondrán algunos de los trabajos que más interés han despertado dentro de la visión por computador.

Leaving Flatland: Toward Real-Time 3D Navigation: (B. Morisset). Este proyecto propone un sistema integral para la localización, el mapeo y la planificación para un robot móvil, utilizado tanto en ambientes interiores como exteriores y basado totalmente en visión 3D. Este sistema de localización basado en odometría visual integra nuevas técnicas de mapeo 3D en tiempo real de los datos obtenidos de un par de cámaras estéreo. El planificador de movimiento utiliza un nuevo enfoque adaptando las técnicas de planificación 2D existentes para luego realizar las operaciones en 3D. Pone a prueba los subsistemas de mapeo y la planificación del movimiento con datos reales y sintetizados para demostrar que tienen un rendimiento computacional favorable para su uso en la navegación autónoma de alta velocidad. En este trabajo implementado con visión estereoscópica utilizan CENSURE, como detector de características y RANSAC para la eliminación de outliers.

Two years of Visual Odometry on the Mars Exploration Rovers: La NASA con sus Mars Exploration Rovers (MER) han demostrado con éxito la capacidad de la odometría visual, poniéndola a prueba en otro planeta por primera vez. Este sistema proporciona a cada rover el conocimiento preciso de su posición, lo que le permite detectar y compensar cualquier deslizamiento imprevisto encontrado durante la traslación de una unidad autónoma. Ha permitido a los rovers moverse de manera segura y con

mayor eficacia en terrenos altamente inclinados y arenosos, lo que conlleva reducir el número de días requeridos para rastrear áreas de interés.

El sistema Odometría Visual MER se basa en un software de detección de imágenes estereoscópicas. Implementa detector de Harris y RANSAC para eliminación de outliers.

Durante estos dos primeros años de operaciones, la Odometría Visual dio un gran salto evolucionando de un sistema adicional a un sistema crítico de seguridad para el vehículo.

Appearance-Guided Monocular Omnidirectional Visual Odometry for Outdoor Ground Vehicles: (Scaramuzza, D. and Siegwart, R). En este trabajo se describe un algoritmo en tiempo real para calcular el movimiento de un vehículo con respecto a la carretera. El algoritmo utiliza como entrada sólo las imágenes proporcionadas por una sola cámara omnidireccional colocada en el techo del vehículo (visión monocular). Scaramuzza propuso un sistema de odometría que usaba una cámara omnidireccional e hizo un gran aporte al proponer un algoritmo que denominaron 1-point RANSAC, el cual permite hacer una estimación más rápida y eficiente de la posición del agente o vehículo al tener en cuenta un modelo de movimiento restringido basado en las restricciones no holonómicas de un vehículo en movimiento.

Visual Odometry (Nister, D; Naroditsky, O.; Bergen, J). Nister en el 2004 propuso uno de los primeros sistemas monoculares que presenta buenos resultados y tolerancia al ruido, este sistema usaba un algoritmo 3D-2D para la estimación de la posición en cada instante y una de sus características más innovadoras está relacionada con el uso del algoritmo 5-point RANSAC para la estimación geométrica del movimiento bajo la presencia de outliers. Este algoritmo obtuvo gran popularidad a partir del trabajo de Nister y ha sido utilizado en varias implementaciones de sistemas monoculares subsiguientes.

SLAM (*Simultaneous Localization and Mapping*): El problema de la Localización y Modelado Simultáneos (SLAM), investiga los problemas que plantea la construcción de modelos matemáticos, geométricos o lógicos de entornos físicos, empleando como herramienta un robot móvil en ocasiones

varios de ellos y el conjunto de sensores y actuadores que lo conforman. Dicho de otra manera, el SLAM busca resolver los problemas que plantea el colocar un robot móvil en un entorno y posición desconocidos, y que él mismo sea capaz de construir incrementalmente un mapa consistente del entorno al tiempo que utiliza dicho mapa para determinar su propia localización. SLAM emplea un filtro extendido de Kalman, siendo esta una de las soluciones más extendidas al problema de la localización y modelado simultáneos, y también es una de las que mejores resultados ha proporcionado en la práctica.

3D Visual Odometry for Road Vehicles (R. Garcia-Garcia • M. A. Sotelo • I. Parra • D. Fernandez • J. E. Naranjo • M. Gavilan) Este trabajo describe un método para estimar la posición global de vehículos en una red de carreteras por medio de odometría visual. Para ello, el movimiento del vehículo en relación con la carretera se calcula utilizando un sistema estéreo-visión montado al lado del espejo retrovisor del coche. El movimiento del vehículo se estima utilizando el enfoque no lineal, basado en RANSAC. Esta técnica iterativa permite la formulación de un método robusto que puede ignorar un gran número de valores atípicos como se encuentran en las escenas de tráfico reales. El método resultante se define como odometría visual y se puede utilizar en conjunción con otros sensores, tales como GPS, para producir estimaciones precisas de la posición global del vehículo. La aplicación obvia del método es proporcionar asistencia al conductor a bordo en las tareas de navegación, o para proporcionar un medio para la navegación autónoma de un vehículo. El método ha sido probado en condiciones de tráfico reales sin necesidad de utilizar el conocimiento previo acerca de la escena ni el movimiento del vehículo.

Capítulo 2

Odometría Visual

2.1 Introducción.

En este capítulo se detalla el funcionamiento del sistema de odometría visual implementado en este trabajo, desglosado en cada una de las partes características de un sistema de visión, partiendo desde la captura de las imágenes hasta la obtención de resultados.

2.2 Descripción del Sistema de Odometría Visual.

El sistema de odometría visual implementado en este proyecto, basado en el estudio de los proyectos detallados en el capítulo de introducción, se compone de las siguientes etapas:

1. Calibración de la cámara.
2. Extracción de características.
3. Emparejamiento de características.
4. Estimación del movimiento.

A continuación se detallará cada una de las distintas alternativas que se pueden implementar en cada una de estas etapas, permitiendo con ello la configuración de diferentes sistemas de odometría.

2.2.1 Calibración de la cámara.

En la actualidad el uso de cámaras digitales está al alcance de cualquiera debido a su bajo precio, pero este abaratamiento de las cámaras actuales trae consigo un problema grave de distorsión. Afortunadamente, este coeficiente de distorsión es constante y mediante métodos de calibración y de reasignación es posible su corrección. Además, mediante la calibración también se puede determinar la relación entre las unidades naturales de la cámara y las unidades del mundo real.

La calibración, como se ha comentado, nos permite extraer información

3D a partir de imágenes 2D capturadas con las cámaras. La matriz de parámetros extrínsecos (matriz esencial) nos permite obtener la proyección 2D sobre el plano de la cámara de las coordenadas 3D de un punto de la escena. Mediante la matriz de parámetros intrínsecos (matriz fundamental) conseguimos las coordenadas 2D en píxeles de la imagen.

El modelo matemático se complica con la inclusión de una serie de coeficientes de distorsión intrínsecos de cada cámara. Lo idóneo es obtener todos estos parámetros una vez y almacenarlos. A partir de los parámetros intrínsecos se obtendrán la matriz de la cámara y los coeficientes de distorsión. La matriz de parámetros extrínsecos (matriz esencial) se puede descomponer en 2 matrices, rotación y traslación ($\boldsymbol{R}$ y $\boldsymbol{t}$).

El sistema de calibración implementado en este proyecto utiliza la librería OpenCV. Para la distorsión OpenCV tiene en cuenta el factor radial y el factor tangencial. Para el factor radial utiliza la siguiente fórmula:

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

Así obtenemos cinco parámetros de distorsión que OpenCV representan como una matriz fila con 5 columnas:

$$Distortion_{coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

Para obtener la conversión de coordenadas de la cámara con coordenadas del mundo real se utiliza la siguiente fórmula:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Los parámetros desconocidos son $\boldsymbol{f_x}$ y $\boldsymbol{f_y}$ (distancia focal de la cámara) y ($\boldsymbol{c_x}$, $\boldsymbol{c_y}$) que son las coordenadas del centro óptico de la cámara expresadas

en píxeles. Si para ambos ejes tienen una distancia focal común y se utiliza una relación de aspecto "***a***" dada (por lo general a=1), entonces:

$$\left[\ \mathbf{f_y = f_x * a}\ \right]$$

Con ello obtenemos en la fórmula superior una sola distancia focal. La matriz que contiene estos cuatro parámetros se conoce como la ***matriz de la cámara.*** Mientras que los coeficientes de distorsión son los mismos independientemente de las resoluciones de cámara utilizados, estos deben ser escalados, junto con la resolución actual de la resolución calibrada.

El proceso de determinar estas dos matrices es la calibración. El cálculo de estos parámetros se realiza a través de ecuaciones geométricas básicas. Las ecuaciones utilizadas dependen de los objetos de calibración elegidos. Actualmente OpenCV es compatible con tres tipos de objetos para la calibración:

- Clásica, tablero de ajedrez en blanco y negro.
- Patrón de círculo simétrico.
- Patrón de círculo asimétrico.

Básicamente, se necesita tomar instantáneas de estos patrones con la cámara y reconocer el patrón capturado. Cada patrón encontrado se traduce en una nueva ecuación. Para resolver la ecuación se necesita al menos un número predeterminado de imágenes del patrón para formar un sistema de ecuaciones bien planteado. Este número es mayor para el patrón de tablero de ajedrez y menor para los patrones de círculo. Por ejemplo, en teoría el patrón de tablero de ajedrez requiere al menos dos instantáneas, sin embargo, en la práctica tenemos una cantidad de ruido presente en nuestras imágenes de entrada bastante alta, por lo que para obtener buenos resultados es probable que se necesite al menos 10 instantáneas del patrón de entrada en diferentes posiciones.

Para llevar a cabo la calibración de la cámara en este proyecto se ha utilizado el patrón de tablero de ajedrez como el que se puede apreciar en la siguiente figura:

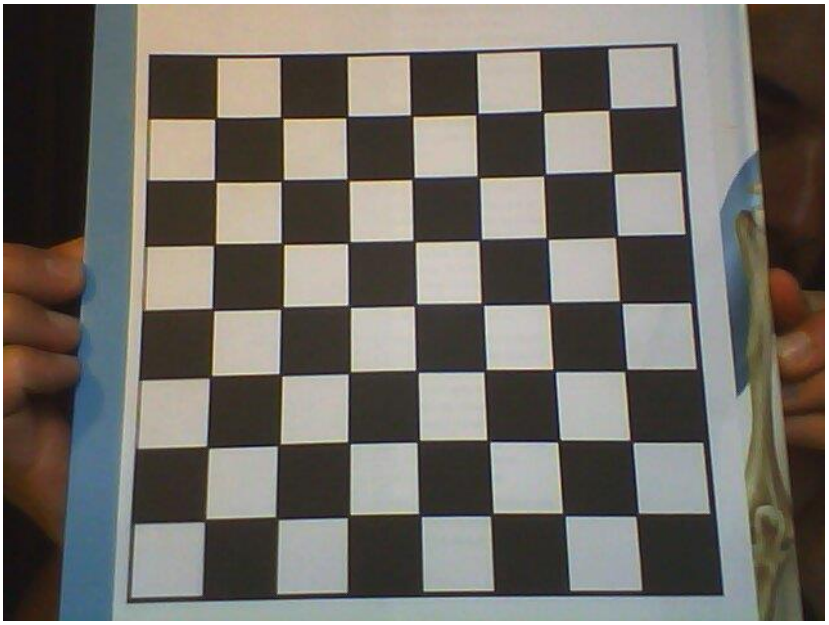

Figura 2: Patrón tablero de ajedrez.

Este tablero debe ser un plano, por tanto es conveniente fijarlo sobre una base plana rígida para que se mantenga plano el interior del patrón.

2.2.2 Extracción de características.

La extracción de características consiste en extraer puntos característicos de una imagen, con el fin de realizar su correspondiente rastreo entre imágenes posteriores y estimar el movimiento.

Actualmente no hay una definición concreta de lo que es una característica, ya que cada definición depende del tipo de aplicación o del sistema que se esté desarrollando. Teniendo en cuenta lo anterior, una característica se define como una parte interesante de una imagen, ya que las características son una parte muy importante para muchos algoritmos de visión por computador. Por este motivo, el algoritmo de detección de características desempeña un papel muy importante dentro de los algoritmos de visión por computador por que dependiendo de lo eficiente y eficaz que sea este algoritmo influirá en la calidad del algoritmo de visión.

La detección de características es una operación de bajo nivel en el procesamiento de imágenes. Esto significa que, por lo general, se realiza como la primera operación en una imagen, se examina cada píxel para ver si hay una

característica presente en él. Si la detección de características forma parte de un algoritmo más complejo, como lo es este sistema de visión, entonces el algoritmo normalmente sólo examina la imagen en la región de las características detectadas y obvia el resto de la imagen al no contener información relevante, con ello conseguimos que el cómputo global del algoritmo principal sea menor. Ocasionalmente, cuando la detección de características es computacionalmente elevada y hay limitaciones de tiempo, es posible la implementación de un algoritmo de nivel superior para ejecutar la etapa de detección de características, de modo que este sólo buscara características en ciertas partes de la imagen.

Prácticamente todos los sistemas de visión por computador utilizan un algoritmo para la extracción de características, siendo este una de las primeras etapas a realizar en el desarrollo del algoritmo. Debido a ello, se han desarrollado un gran número de detectores de características. Estos varían ampliamente según el tipo de característica detectada, la complejidad computacional y la repetibilidad. Estos algoritmos los podemos clasificar en tres grandes grupos:

- Detectores de regiones. (Blobs detectors).
- Detectores de esquinas. (Corners detectors).
- Detectores de bordes. (Edge detectors).

2.2.2.1 Detectores de Blobs (Detector de regiones).

En la visión artificial, los detectores de blobs se caracterizan por las técnicas implementadas para detectar puntos o regiones más claras o más oscuras de la imagen. Hay dos tipos de detectores de blobs, los que usan métodos diferenciales y los que implementan métodos basados en extremos locales. Estos detectores también se denominan detectores de puntos de interés, o detectores de regiones interesantes.

El desarrollo y la implementación de este tipo de detectores es bastante relevante ya que aportan información sobre regiones que otros detectores no pueden detectar. Estos detectores se utilizan como una etapa fundamental para posteriormente realizar el reconocimiento o seguimiento de objetos. Otro uso habitual de estos detectores tiene que ver con el análisis de texturas y su reconocimiento.

Las técnicas implementadas en estos detectores se utilizan en aplicaciones que podemos usar en nuestra vida cotidiana como por ejemplo para software de dispositivos táctiles, funciones de detección de rostros y sonrisas en cámaras de fotos, sistemas de vigilancia y seguridad, o para analizar imágenes médicas (Diagnóstico Asistido por Ordenador).

Dentro de los detectores de Blobs podemos destacar los siguientes algoritmos:

- SIFT. (Scale-Invariant Feature Transform)
- SURF. (Speeded-Up Robust Features)
- CENSURE.(Center Surround Extremas).

2.2.2.2 Detectores de esquinas (Corners Detectors).

Los detectores de esquinas son uno de los métodos más utilizados en el campo de la visión artificial para la obtención de puntos característicos, ya que, demuestran una gran repetibilidad siendo está una característica fundamental y bastante importante en un buen detector de características.

Una esquina puede definirse como la intersección de dos líneas que forman un ángulo. También puede definirse como la arista o el ángulo que resulta del encuentro de dos superficies, principalmente la de las paredes de un edificio.

En este tipo de detectores un punto de interés es un punto en una imagen que tiene una posición bien definida y puede ser detectado de forma robusta. Por ello, un punto de interés puede ser una esquina pero también puede ser, por ejemplo, un punto aislado de intensidad local máxima o mínima, el final de una línea, o un punto en una curva donde la curvatura es localmente máxima.

Los detectores de esquinas normalmente no son muy robustos y a menudo requieren de una posterior comprobación o la implementación de métodos de repetibilidad para reducir el número de errores en la tarea de reconocimiento.

Una forma de determinar la calidad de un detector de esquinas es su capacidad para descubrir la misma esquina en múltiples imágenes similares, bajo condiciones de iluminación diferentes, traslación y rotación entre otras transformaciones. Una implementación simple para la detección de esquinas

en imágenes es usando la correlación, es decir, comparar que una característica en un frame anterior se corresponde con otra característica de un frame posterior si ambas responde al mismo comportamiento en función de la variación de sus parámetros, pero esto es costoso computacionalmente y poco óptimo. Otra implementación bastante usada está basada en un método propuesto por Harris y Stephens, que a su vez es una mejora del algoritmo de Moravec.

Algunos de los algoritmos que implementa detector de esquinas y que ha día de hoy son más utilizados son:

- Harris Corner Detection.
- Shi-Tomasi Corner Detector
- FAST (Features from Accelerated Segment Test)

2.2.2.3 Detectores de bordes. (Edge Detectors).

Los detectores de bordes son algoritmos de detección de características que se utilizan con menor frecuencia en implementaciones de sistemas de visión artificial. Este tipo de detectores trata de localizar puntos en una imagen en la que esta cambia drásticamente siendo uno de los principales causantes de este cambio brusco el brillo de la imagen en cada uno de los pixeles.

La detección de estos cambios brusco de brillo se pueden deber a cambios en el escenario que se está capturando, movimiento de objetos, giros de la cámara, etc.

En el caso ideal, los resultados de aplicar un detector de bordes a una imagen pueden conducir a un conjunto de curvas conectadas que indican las fronteras de los objetos, las fronteras de las marcas en las superficies de estos objetos y curvas que corresponden a discontinuidades en la orientación de las superficies.

Al aplicar un algoritmo de detección de bordes a una imagen es posible reducir la cantidad de datos a ser procesados, por lo que se puede filtrar la información que puede ser considerada como menos relevante, logrando preservar las propiedades estructurales de la imagen. Si el paso de detección de bordes fue satisfactorio, el paso de interpretar el contenido en la imagen original se puede reducir sustancialmente. Sin embargo, no siempre se obtienen esos bordes ideales a partir de imágenes reales, con una complejidad moderada.

La detección de bordes es uno de los pasos fundamentales en el procesamiento de imágenes, en el análisis de imágenes, en el reconocimiento de patrones en una imagen, y en las técnicas de visión por computador. Sin embargo, en años recientes (2010) se han realizado investigaciones en métodos de visión por computadora que no dependen explícitamente en la detección de bordes como un paso de preprocesamiento.

Ejemplos de detección de bordes los podemos ver en diferentes aplicaciones que usamos habitualmente, un ejemplo claro es una herramienta presente en los editores fotográficos (Photoshop, Gimp,...) para realizar la selección de formas concretas dentro de una imagen.

Otro ejemplo de este tipo de algoritmo lo tenemos en la librería de OpenCV:

- Canny Edge Detector.

2.2.3 Emparejamiento de características

El emparejamiento de características es el proceso mediante el cual se deben establecer relaciones entre las características obtenidas de una imagen o frame I_K (imagen posterior) con las correspondientes características obtenidas de una imagen o frame I_{K-1} (imagen anterior). Es decir, establecer relaciones entre las características extraídas en imágenes tomadas en diferente momento. Este proceso se le conoce como emparejamiento de características o Feature Matching.

Dependiendo de la técnica usada para el emparejamiento de características obtendremos un algoritmo más o menos eficiente computacionalmente. La manera más simple de establecer relaciones entre características es ir comparando todas y cada una de los puntos característicos obtenidos en una imagen con todos los puntos de interés obtenidos de la imagen anterior. A este tipo de emparejamiento se le conoce como emparejamiento no restringido o unconstrained matching. En cambio, si se realiza un emparejamiento basado en emparejamiento restringido o constrained matching, obtenemos un algoritmo mucho más eficiente computacionalmente, debido a que este tipo de emparejamiento utiliza tablas hash o arboles de búsqueda para realizar el emparejamiento, reduciendo considerablemente el número de comparaciones a realizar.

En OpenCV tenemos a nuestra disposición dos algoritmos de

emparejamientos, **Brute Force Matcher** y **FLANN**, el primero de ellos corresponde a un emparejamiento no restringido y el segundo a un emparejamiento restringido.

Brute Force Matcher es bastante simple. Selecciona un descriptor de características del primer frame o fotograma obtenidos de la fase de extracción y lo compara con cada uno de los descriptores de características del segundo frame, tratando de establecer una relación entre ellos basándose en alguna de las medidas de similitud que establecen relación calculando la distancia y aquella más cercana será la que devuelva. Entre algunas de estas medidas de similitud se encuentran SSD o SAD(Sum of Square Differences), NCC (Normal Cross Correlation) NNDR(Nearest Neighbor distance Radio). Estas medidas de similitud hacen parte de la primera etapa del Feature Matching y son usadas por muchas implementaciones.

FLANN significa Biblioteca rápida de Vecinos aproximados más cercanos. Utiliza la medida de similitud NNDR(Nearest Neighbor distance Radio). Contiene una colección de algoritmos optimizados para la búsqueda rápida del vecino más cercano en grandes conjuntos de datos y para las características de altas dimensiones. Para ello utiliza árboles de búsquedas con lo que su ejecución es mucho más rápida tanto es así que funciona más rápido que Brute Force Matcher para grandes conjuntos de datos.

2.2.4 Estimación del movimiento.

La estimación de movimiento es el proceso de determinar los vectores de movimiento que describen la transformación de una imagen a otra, para ello se utilizaran las características obtenidas en las etapas de extracción y emparejamiento de características extraídas de un fotograma o frame anterior y otro posterior. Es un problema bastante complejo de resolver debido a que el movimiento es en tres dimensiones y las imágenes son una proyección de la escena 3D sobre un plano 2D. Hay diversas metodologías que plantean el problema dependiendo de cómo se hayan obtenido esas características, bien haciendo relación entre características representadas ambas en las mismas dimensiones (metodologías 3D-3D o 2D-2D), o en dimensiones diferentes (metodología 3D-2D). Esta etapa es fundamental en un sistema de visión por computador, esto se debe a que la trayectoria total del vehículo y su posición

se pueden obtener concatenando las estimaciones de movimiento de cada par de imágenes capturadas.

El procedimiento a seguir es calcular la geometría de la cámara para obtener la rotación y la traslación usando los puntos característicos emparejados. Todos los cálculos que se realizan en este paso están basados en el concepto de geometría epipolar, por lo tanto la primera tarea es calcular las matrices fundamentales que existen para cada par de fotogramas consecutivos.

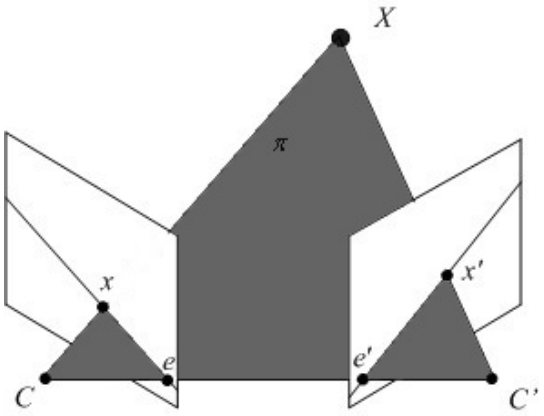

Figura 3: Geometría epipolar.

2.2.4.1 Cálculo de la matriz fundamental

Para calcular la matriz fundamental se ha utilizado el algoritmo de ocho puntos (*Hartley, R.~I. and Zisserman, A.)* aplicando RANSAC para la eliminación de outliers, gracias a la librería OpenCV se puede realizar mediante el método *cv2.**findFundamentalMat()*** en el cual podemos seleccionar diferentes algoritmos para la eliminación de outliers. El algoritmo se basa en la restricción epipolar para construir un sistema de ecuaciones lineal que cuyo resultado es la matriz fundamental:

$$\mathbf{X'\,F\,x = 0}$$

siendo **F** la matriz fundamental, x las coordenadas de una características en un fotograma y x' las coordenadas de esa misma característica en el siguiente fotograma. La matriz **F** es de ocho grados de libertad, por lo tanto son necesarios un mínimo de ocho puntos emparejados para que el sistema quede totalmente determinado.

Una vez obtenida la matriz fundamental procedemos a calcular la matriz esencial, para ello utilizaremos también la matriz de calibración de la

cámara.

2.2.4.2 Calculo de la matriz esencial

Al realizar la transformación de calibración para cada uno de los dos puntos, lo que relaciona el punto calibrado de una imagen con el punto calibrado una imagen posterior es la composición de la translación y la rotación de las vistas. Por ello, podemos considerar a la matriz fundamental como la matriz que relaciona dos proyecciones, es decir, que relaciona las dos imágenes de una misma escena mientras que la matriz esencial **E** relaciona el movimiento relativo para llegar de una vista a la otra vista.

Una vez se conoce la matriz fundamental, es posible calcular una reconstrucción 3D de la escena. Si se conocen los parámetros intrínsecos de la cámara, se puede calcular la matriz esencial **E** a partir de la matriz fundamental:

$$\mathbf{E} = \mathbf{K'}^T \mathbf{F}\ \mathbf{K}$$

donde **K** es la matriz de calibración de la cámara en cuestión.

El siguiente paso a realizar es calcular las matrices de la cámara en cada uno de los fotogramas.

2.2.4.3 Calculo de matrices de la cámara, matriz de rotación y vector traslación.

Si asumimos que la imagen **I** está asociada a una cámara representada por la matriz **P**, mientras que la imagen **I'** a una cámara **P'**, lo que queremos obtener son justamente estas dos matrices. De hecho, si tomamos a **P** como la cámara canónica,

$$\mathbf{P} = \mathbf{K}[\mathbf{I}|\mathbf{0}]$$

la incógnita es únicamente **P'**. Notar que, si bien tenemos dos matrices de cámara distintas, ambas comparten los mismos parámetros intrínsecos representados por **K**. Mediante ciertos análisis teóricos es posible ver que, a partir de la información que se tiene, existen cuatro posibilidades para la matriz **P'** dado que existen ciertas ambigüedades. Éstas, surgen a partir del

hecho de que hay dos posibles rotaciones y dos posibles traslaciones:

$$R_1 = U\,W^T\,V^T$$
$$R_2 = U\,W\,V^T$$
$$T_1 = T$$
$$T_2 = -T$$

donde:

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

y las matrices **U**, **V** son el resultado de la descomposición **SVD** de **E**:

$$\mathbf{E = UDV^T}$$

En la práctica, en caso de que $\det(\mathbf{V^T}) < 0$ o $\det(\mathbf{U}) < 0$, será necesario tomar la descomposición **SVD** de $-\mathbf{E}$ en realidad. Finalmente, el vector **T** de traslación se puede obtener resolviendo el sistema homogéneo: $\mathbf{ET} = 0$ Otra posibilidad para encontrar **T** es tomar la tercera columna de **U**:

$$\mathbf{T = u_3}$$

La demostración de esta afirmación se puede consultar en (Hartley & Zisserman, 2000).

A partir de las cuatro posibilidades para **P'**, que surgen de hacer todas las combinaciones de $\mathbf{R_1}$, $\mathbf{R_2}$ y $\mathbf{T_1}$, $\mathbf{T_2}$ posibles, habrá solo una de ellas que será la correcta.

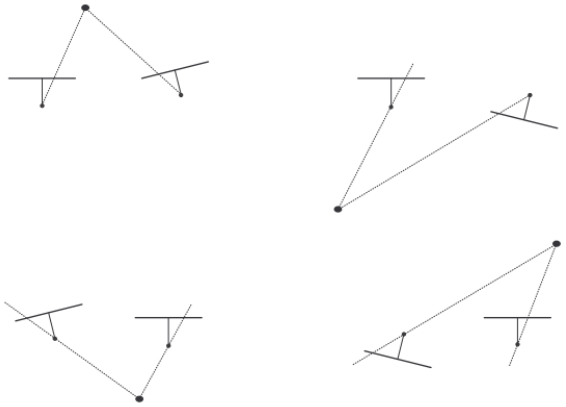

Figura 4: Imagen de las cuatro posibles soluciones para la segunda cámara.

Esto se debe al hecho de que tomaremos aquella en donde los puntos que observamos de la escena queden delante del plano de la imagen. Tomando entonces algún par de correspondencias de las obtenidas previamente, habrá que obtener la posición **Q** en el mundo del punto asociado a las mismas. Para obtener dicha posición se hace un simple método de triangulación a partir de las cámaras **P** y la **P'** considerada en cada caso. De esta forma, entonces, obtendremos cuatro posibles valores para **Q**. Lo que resta, es determinar entonces si dicho punto queda delante o no del plano de la imagen de ambas cámaras. Se puede ver que si se cumple que:

$$\mathbf{QzQw > 0 \ y \ (P'Qz)Qw > 0}$$

entonces el punto queda delante de ambos planos de imagen.

Por consecuencia, la **P'** utilizada para obtener el **Q** en cuestión es la cámara correcta.

Finalmente, para obtener la rotación y traslación existente entre las cámara **P** y **P'**, simplemente se toman la rotación y traslación utilizadas para construir **P'**.

Capítulo 3

Diseño e implementación.

3.1 Introducción

En este capítulo se expondrá una explicación del algoritmo usado en cada etapa, con algunos detalles de implementación importantes para asegurar el funcionamiento del sistema de odometría visual.

3.2 Arquitectura general del sistema

Para la implementación de este sistema de odometría visual se ha utilizado ampliamente la librería OpenCV. OpenCV es una librería open-source (código abierto) de visión artificial originalmente desarrollada por Intel. Esta librería contiene múltiples funciones que son de utilidad en múltiples aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos.

Las características principales que hacen de OpenCV son:

- Librería de código abierto publicada bajo licencia BSD, con la posibilidad de ser utilizada tanto para fines comerciales como para investigación, siempre y cuando se respetan las condiciones especificadas en ella.
- Es una librería multiplataforma, OpenCV tiene diferentes versiones adaptadas a sistemas operativos tales como GNU/Linux, Mac OS X o Windows.
- Como no podía ser de otra forma también se puede implementar su uso para sistemas operativos móviles, como Android y IOS.
- Posee más de 500 funciones que abarcan múltiples áreas en procesos visión, desde reconocimiento de objetos, calibración de cámaras, reconocimiento facial y un largo etc.
- Su fácil implementación y sus resultados altamente eficientes.

- Desarrollada en C y C++, siendo, además, compatible con el lenguaje de programación Python.

Todo esto hace de OpenCV una de las librerías más utilizadas en los diferentes sistemas de visión por computador.

La implementación no está orientada a objetos y ha sido desarrollada en el lenguaje de programación Python por la facilidad y sencillez a la hora de usar la librería OpenCV.

A continuación se mencionan los módulos utilizados de esta librería.

Módulos OpenCV:

- cv2.imread().
- cv2.cvtColor().
- cv2.findChessboardCorners().
- cv2.cornerSubPix().
- cv2.drawChessboardCorners().
- cv2.imshow().
- cv2.calibrateCamera().
- cv2.getOptimalNewCameraMatrix().
- cv2::SurfDetectorAndCompute.
- cv2.BFMatcher().
- cv2**:**FlannBasedMatcher.
- cv2.findFundamentalMat()
- cv2.SVDecomp()
- cv2.triangulatePoints()
- cv2::DMatch.
- cv2::KeyPoint.

Cada uno de los módulos será detallado en las siguientes secciones siguiendo la arquitectura del sistema de odometría visual.

3.3 Captura de imágenes

Para la obtención de las imágenes se utilizó una cámara digital de alta definición, concretamente una SJCAM modelo SJ4000, cuyas características técnicas son:

- Pantalla: 1,5 pulgadas
- Lente: Objetivo gran angular de 170 ° A + HD
- Idioma: Inglés / alemán / francés / español / italiano / portugués / chino tradicional / chino simplificado / japonés / ruso
- Resolución:1080P 30FPS (1920 x 1080) / 720p (1280 x 720) 60FPS / VGA (848 x 480) 60 FPS / QVGA (640 x 480) 60 FPS
- Formato de vídeo: MOV
- Formato de Videos: H.264
- Resolución de las fotos: 12M / 8M / 5 M
- Almacenamiento: MicroSD
- Frecuencia de Fuente óptico: 50Hz / 60Hz
- Interfaz USB: USB2.0
- Interfaz Fuente de alimentación: 5V1A
- Capacidad de la batería: 900mAh
- Potencia de disipación: 4.2V 400mA
- Tiempo de grabación: 1080P / 70 minutos
- Tiempo de carga: 3 horas
- Sistema Operativo (SO): Sistema operativo Windows XP/Vista/Win7/ Windows 8 / Mac
- Dimensión: 59.27 mm * 41.13 mm * 29.28 mm.

La cámara fue montada en el techo de un Ford Fiesta Trend, a una altura de 1,60 metros, orientada con un ángulo de cero grados con respecto a la carretera tal y como se puede apreciar en la imagen.

Figura 5: Montaje de cámara sobre el techo del *vehículo.*

Se captaron diversas secuencias con diferentes giros y movimientos para la prueba del algoritmo. En este proceso de captación de secuencias de imágenes surgieron diversos problemas (tráfico, deslumbramiento del sol, peatones,...) debido a ello, fueron necesarias obtener bastantes secuencias para luego poder seleccionar aquellas en las que se obtuvo unas imágenes optimas ya que este algoritmo no está optimizado para distinguir objetos que se pueda encontrar durante la captación de imágenes cuyo movimiento pueda causar errores en la detección del flujo óptico.

Posteriormente, se procesaron las imágenes obtenidas para disminuir el número de fotogramas por segundo que tenía cada secuencia. Este preprocesamiento de las secuencias se realiza debido a que la cantidad de fotogramas por segundo que nos capta la cámara utilizada es demasiado elevada para obtener un movimiento correcto entre los puntos característicos detectados entre el fotograma anterior y el posterior, ya que al realizar la toma de imágenes a una velocidad reducida además de captar un elevado número de fotogramas por segundo y el error en el movimiento añadido a las imágenes producido por las vibraciones del vehículo, hacen que el movimiento detectado entre los puntos característicos entre un fotograma posterior y uno anterior pudiese ser erróneo. Con la disminución en el número de fotogramas también se consigue mejorar el tiempo de ejecución del algoritmo ya que este emplea entre 1 y 1,5 segundos en procesar cada fotograma. Este procedimiento también se podría realizar en el propio algoritmo de visión seleccionando un número determinado de frames, pero para mejorar la velocidad de computo del algoritmo se decidido editar los videos para evitar cargar y leer frames innecesarios en la ejecución del mismo.

3.4 Calibración

Las cámaras pinhole económicas de hoy presenta una gran cantidad de distorsión en las imágenes. Dos importantes distorsiones son la distorsión radial y distorsión tangencial. Debido a la distorsión radial, las líneas rectas aparecerán curvadas. Su efecto es mayor a medida que nos alejamos del centro de la imagen. Gracias a la librería OpenCV y al módulo cv2:CameraCalibrate se ha realizado la calibración de la cámara para poder corregir estas distorsio-

nes en las imágenes. Como ejemplo, se muestra dos imágenes obtenidas la primera antes de la calibración (Figura 15) y la segunda después de ser calibrada (Figura 16).

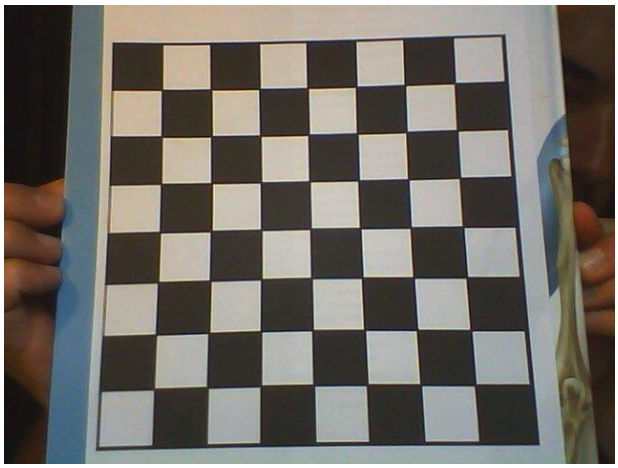

Figura 6: Sin calibración

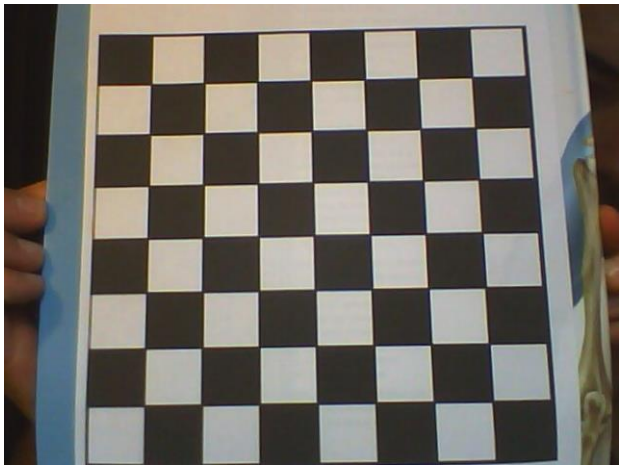

Figura 7: Calibrada.

El módulo de calibración de la cámara sigue el siguiente esquema:

1. Declaramos las variables necesarias para el algoritmos (object points, imagen points,...).
2. Cargamos la imagen con el método *cv2.imread()* de OpenCV.
3. Convertimos la imagen a escala de grises utilizando *cv2.cvtColor().*
4. Utilizamos el método *cv2.findChessboardCorners()* al que le pasamos la imagen en escala de grises para localizar las esquinas del tablero de ajedrez.
5. Mediante *cv2.cornerSubPix()* seleccionamos las esquinas del tablero para posteriormente dibujarlas.
6. Para dibujar en la imagen las esquinas seleccionadas del tablero utilizamos el método *cv2.drawChessboardCorners().*
7. Mostramos la imagen para verificar que el proceso se ha realizado con éxito, para ello utilizamos el módulo *cv2.imshow().*
8. Una vez comprobado los resultados pasamos a realizar la primera calibración de la cámara, para ello se utiliza el módulo *cv2.calibrateCamera()* al que le pasamos los object points, imagen points y la imagen en escala de grises, para obtener una primera matriz de calibración.

9. Ya por último, para obtener la matriz de calibración optima se utiliza el módulo *cv2.getOptimalNewCameraMatrix()* al que le pasaremos la anterior matriz obtenida y nos retornara la matriz de calibración optimizada.

3.5 Extracción de características

En esta etapa se han implementado varios algoritmos de extracción de características, debido a que su implementación se realiza de una manera bastante rápida y sencilla por ello se han probado varios de los módulos que disponemos en la librería OpenCV para comprobar el comportamiento del sistema de visión con cada uno de ellos, (velocidad de ejecución, selección de características,...).
El primer de los algoritmos implementados fue el detector de esquinas de Harris. OpenCV tiene la función *cv2.cornerHarris()* para este propósito.

Sus argumentos son:

- **Img**: imagen de entrada, debe ser en escala de grises y tipo flotante.
- **BlockSize**: Es el tamaño del bloque considerado para la detección de esquina.
- **Ksize**: parámetro apertura.
- **K**: parámetro libre en la ecuación del detector de Harris.

Se decidió comenzar con el detector de esquinas de Harris ya que investigadores como Nister y Scaramuzza respaldan el uso de este detector para el desarrollo de un sistema de odometría visual. Este detector género un número considerable de puntos de interés detectados pero a la hora de implementar el emparejamiento de características no se obtenían unos resultados óptimos de emparejamientos, se trató de ajustar el detector con diferentes combinaciones en los parámetros del mismo para tratar de conseguir mejores resultados sin llegar a tener éxito.

En segundo lugar se implementó el detector de blobs conocido como SIFT, es un detector ampliamente usado en la visión por computador por sus buenos resultados al detectar puntos de interés.

Para usar SIFT en OpenCV primero tenemos que crear un objeto SIFT mediante *cv2.sift()*.Podemos pasar distintos parámetros a la misma que son opcionales siendo estos:

Parametros:

- **nfeatures**: Número de las mejores características para retener. Las características se clasifican por sus puntuaciones (medidos en el algoritmo SIFT como el contraste local)
- **nOctaveLayers**: Número de capas en cada octava. El número de octavas se calcula automáticamente mediante la resolución de la imagen.
- **ContrastThreshold**: El umbral de contraste utilizado para filtrar las características débiles (bajo contraste) en regiones semi-uniformes. Cuanto mayor sea el umbral, serán menos los puntos de interés reconocidos por el detector.
- **EdgeThreshold**: El umbral utilizado para filtrar las características similares. Hay que tener en cuenta que su significado es diferente de la contrastThreshold, es decir, cuanto mayor sea el edgeThreshold, mayor serán los puntos de interés retenidos.
- **Sigma**: Valor para la función gaussiana que se aplica a la imagen de entrada. Si la imagen es capturada con una cámara económica quizás sea conveniente reducir el número sigma para un mejor resultado.

Con este detector se obtuvieron mejores resultados que con el detector de esquinas de Harris en la etapa de emparejamiento de características.

Por último y como algoritmo de detección de características elegido se implementó SURF. Surf como ya hemos mencionado es un detector de blobs al igual que SIFT, la gran ventaja que tiene SURF sobre SIFT es que este es más rápido en su ejecución obteniendo prácticamente los mismos resultados que SIFT.

Para usar SURF en OpenCV primero tenemos que crear un objeto SURF mediante cv2.surf().Podemos pasar distintos parámetros a la misma:

Parámetros:

- **HessianThreshold**: Umbral para el detector de punto clave utilizados en SURF.

- **nOctaves**: Número de octavas que el detector de puntos clave utilizará.
- **nOctaveLayers**: Número de láminas de octava dentro de cada octava.
- **Extended**: Especifica el tamaño de los descriptores (true - uso extendido descriptores de 128 elementos; falsas - utilizan descriptores de 64 elementos).
- **Upright**: Especifica si detecta orientación de los puntos de interés (verdadero - no calcula la orientación de características, y false - calcula la orientación).

Una vez hemos creado el objeto SURF podemos obtener los descriptores y los puntos de interés con el modulo *surf.detectAndCompute(),* al que le pasaremos la imagen en escala de grises.

3.6 Emparejamiento de características

En la etapa de emparejamiento de características se han probado dos algortimos que ofrece la librería OpenCV cuya metodología es totalmente diferente, BruteForceMatcher y FLANN.

En una primera implementación se utilizó BruteForceMatcher, para ello creamos un objeto *BFMatcher().* Luego usamos el método *Matcher.match()* para obtener los mejores emparejamientos entre las dos imágenes. Las ordenamos ascendentemente por valor de sus distancias de manera que los resultados más importantes (con baja distancia) los tenemos en los primeros valores.El resultado de *Matcher.match()* es una lista de objetos DMatch.

En una segunda implementación para el emparejamiento de características se ha decidido usar el cv2**:**FlannBasedMatcher debido a que fue el que mejores resultados ofrecía en cuestión de tiempo de ejecución. (FLANN: biblioteca rápida de vecinos aproximados más cercanos). Contiene una colección de algoritmos optimizados para la búsqueda rápida del vecino más cercano en grandes conjuntos de datos y para las características de altas dimensiones. Funciona más rápido que BFMatcher para grandes conjuntos de datos.

3.7 Estimación del movimiento

Para la estimación del movimiento el procedimiento a seguir fue calcular la geometría de la cámara para obtener la rotación y la traslación usando los puntos característicos emparejados. Todos los cálculos que se realizaron en este paso están basados en el concepto de geometría epipolar, por lo tanto la primera tarea es calcular las matrices fundamentales que existen para cada par de fotogramas consecutivos.

Para el cálculo de la matriz fundamental se utilizó el módulo de OpenCV, *cv2.findFundamentalMat(points1,points2,cv.CV_FM_RANSAC, param1, param2)* que nos permite seleccionar diferentes algoritmos para la obtención de la matriz.

Los parámetros que le podemos pasar son:

- ***Points1***: Array de N puntos de la primera imagen. Las coordenadas de los puntos deben estar en coma flotante (simple o doble precisión).
- ***Points2***: Array de N puntos de la segunda imagen. Las coordenadas de los puntos deben estar en coma flotante (simple o doble precisión).
- Método para calcular una matriz fundamental:
  - ***CV_FM_7POINT*** para un algoritmo de 7 puntos.
  - ***CV_FM_8POINT*** para un algoritmo de 8 puntos.
  - ***CV_FM_RANSAC*** para el algoritmo RANSAC.
  - ***CV_FM_LMEDS*** para el algoritmo LMedS.
- ***param1***: El parámetro utilizado por RANSAC. Es la distancia máxima desde un punto a una línea epipolar en píxeles, más allá de este el punto se considera un valor atípico y no se utiliza para el cálculo de la matriz fundamental final. Se puede ajustar entre los valores 1 a 3, dependiendo de la precisión de la localización del punto, resolución de imagen, y el ruido de la imagen.
- ***param2***:Parámetro usado únicamente para los métodos RANSAC o LMedS. Se especifica un nivel deseable de confianza (probabilidad) de que la matriz estimada es correcta.

Una vez calculada la matriz fundamental se calcula la matriz esencial **E** para posteriormente hacer una descomposición SVD y obtener de ahí la

matriz de rotación y el vector de traslación.

Para el cálculo de la matriz esencial se utilizado también la matriz de calibración de la cámara y su traspuesta:

$$\mathbf{E = K^T\ F\ K}$$

Una vez obtenida la matriz esencial se realiza la descomposición mediante SVD, para ello se utilizó el módulo *cv2.SVDecomp(Essential),* obteniendo de ello las matrices U,V y W, que se utilizarán para el cálculo de la rotación y la traslación.

Después de calcular las diferentes matrices de rotación y traslación se debe comprobar cuál de las posibles combinaciones es la correcta, para ello se realiza una triangulación de los puntos obtenidos con la matriz de la cámara para saber que combinación dibuja los puntos al frente de ambas cámaras y así obtener la rotación y traslación correcta. Para ello se utiliza el modulo *cv2.triangulatePoints(P, P_2, points1, points2)* con el que obtenemos una representación de los puntos en las coordenadas del mundo, pudiendo así evaluar cuál de las posibles combinaciones de las matrices de rotación y traslación es la correcta.

Capítulo 4

Resultados

4.1 Introducción

La puesta en práctica del algoritmo ha sido especialmente delicada. Se han realizado numerosas tomas de datos con distintas problemáticas, que han hecho tener que volver a repetirlas en numerosas ocasiones. A continuación se presentan los resultados obtenidos por el sistema de odometría visual propuesto.

4.2 Prueba línea recta.

Esta secuencia fue capturada en un ambiente de carretera y se compone de 586 fotogramas que representan 19 segundos de recorrido (192 fotogramas después de editado) con una ejecución de 201 segundos. Para esta prueba es importante mencionar que el vehículo se desplaza a una velocidad prácticamente constante de unos 20 Km/h.

En la figura 8 se puede observar la trayectoria del vehículo en una visión aérea extraída de google Maps.

Figura 8: Plano de los exteriores donde se ha realizado la primera prueba de trayectoria en línea recta.

La grafica obtenida de la secuencia de fotogramas extraída del video refleja el resultado obtenido para esta prueba:

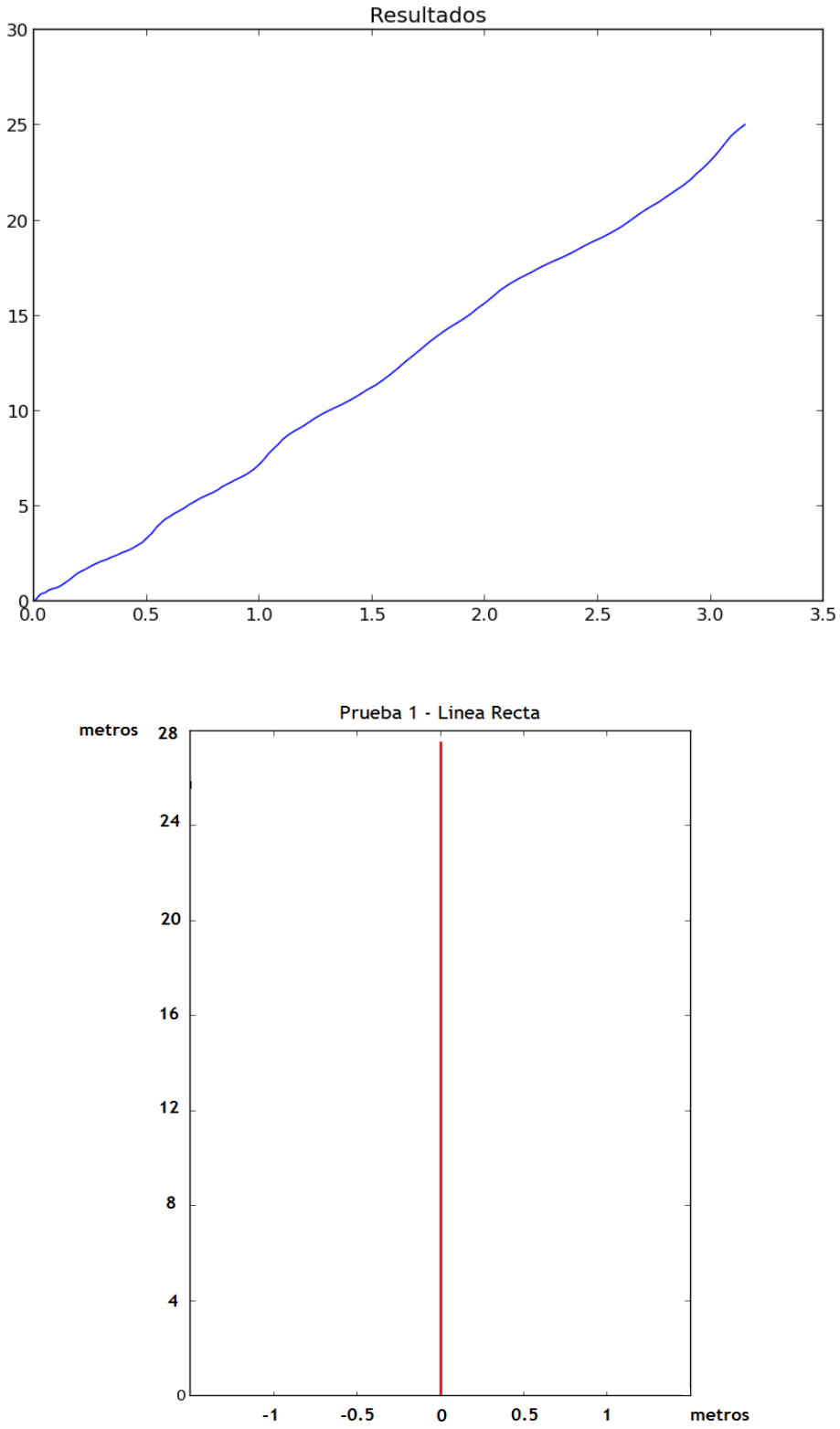

Figura 9: Graficas resultados, sistema (azul), ground truth (rojo).

La tabla siguiente muestra la diferencia entre la estimación del sistema de odometría visual diseñado y el Ground Truth (G.T.) para ciertos fotogramas de la secuencia de imágenes de esta prueba. El error (D) se muestra en metros y es calculado como la distancia euclidiana entre la estimación y el Ground Truth.

Instante	X (G.T.)	Y (G.T.)	X (estimación)	Y (estimación)	Error (D)
0	0,0	0.0	0,0	0,0	0,0
1	0,0	4.358	0,434	3,489	0,94566908
2	0,0	8.035	1,123	6,989	1,37080706
3	0,0	12.297	1,489	10,622	2,31983836
4	0,0	16.325	2,008	14,899	2,28935733
5	0,0	19.966	2,578	17,687	3,18294282
6	0,0	24.352	3,190	22,315	3,71103395
Final	0,0	27.854	3,23	25	4,14358927

Tabla 1: Estimación del sistema de odometría visual diseñado y el Ground Truth para trayectoria recta.

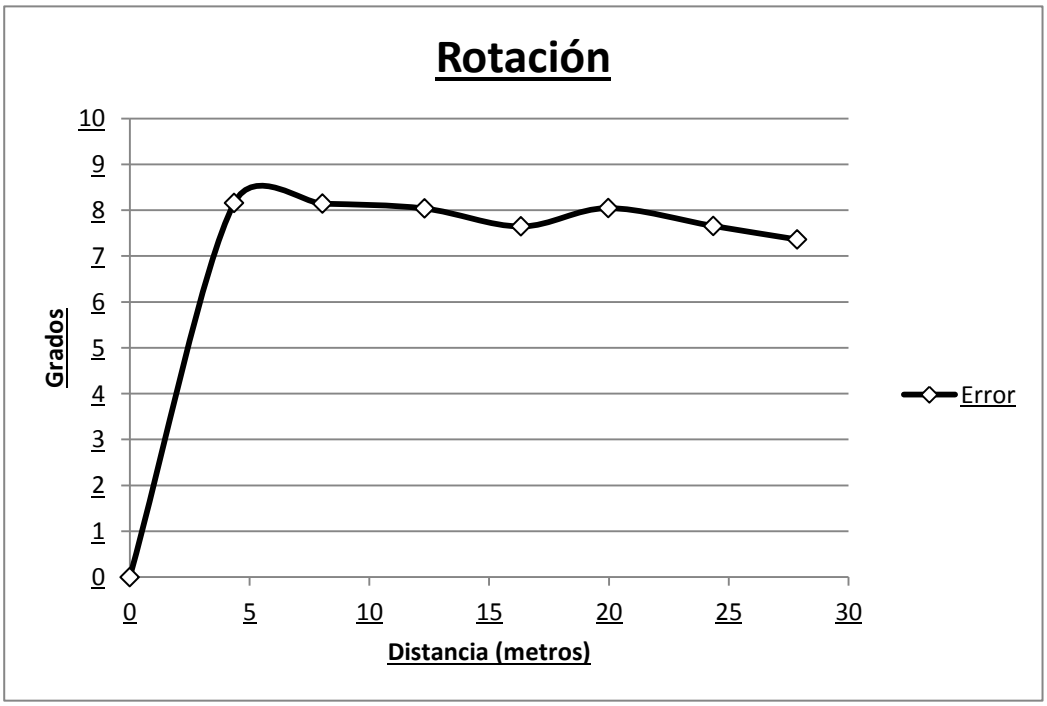

Figura 10: Error de rotación en función de la trayectoria. Prueba trayectoria recta.

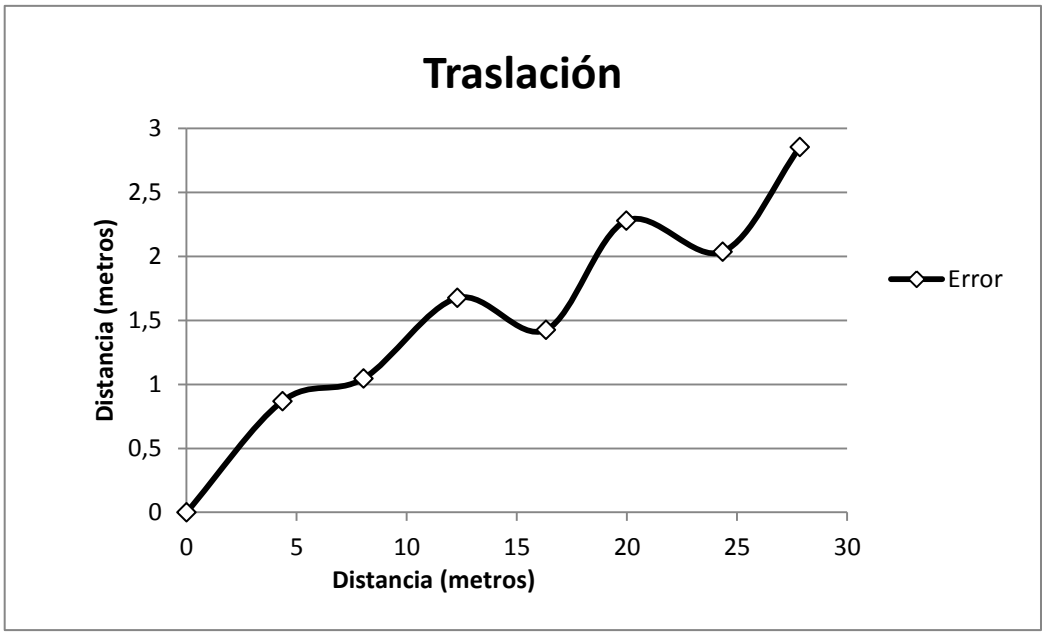

Figura 11: Error de traslación en función de la trayectoria. Prueba trayectoria recta.

4.3 Prueba giro derecha.

La secuencia fue capturada en un ambiente de carretera y se compone de 664 fotogramas que representan 22 segundos de recorrido (221 fotogramas después de editado) con un tiempo de ejecución de 231 segundos. Para esta prueba es importante mencionar que el vehículo se desplaza a una velocidad prácticamente constante de unos 20 Km/h.

En la figura 12 se puede observar la trayectoria del vehículo.

Figura 12: Plano de los exteriores donde se ha realizado la segunda prueba de trayectoria con giro hacia la derecha.

La grafica obtenida de la secuencia de fotogramas extraída del video refleja el resultado para esta prueba:

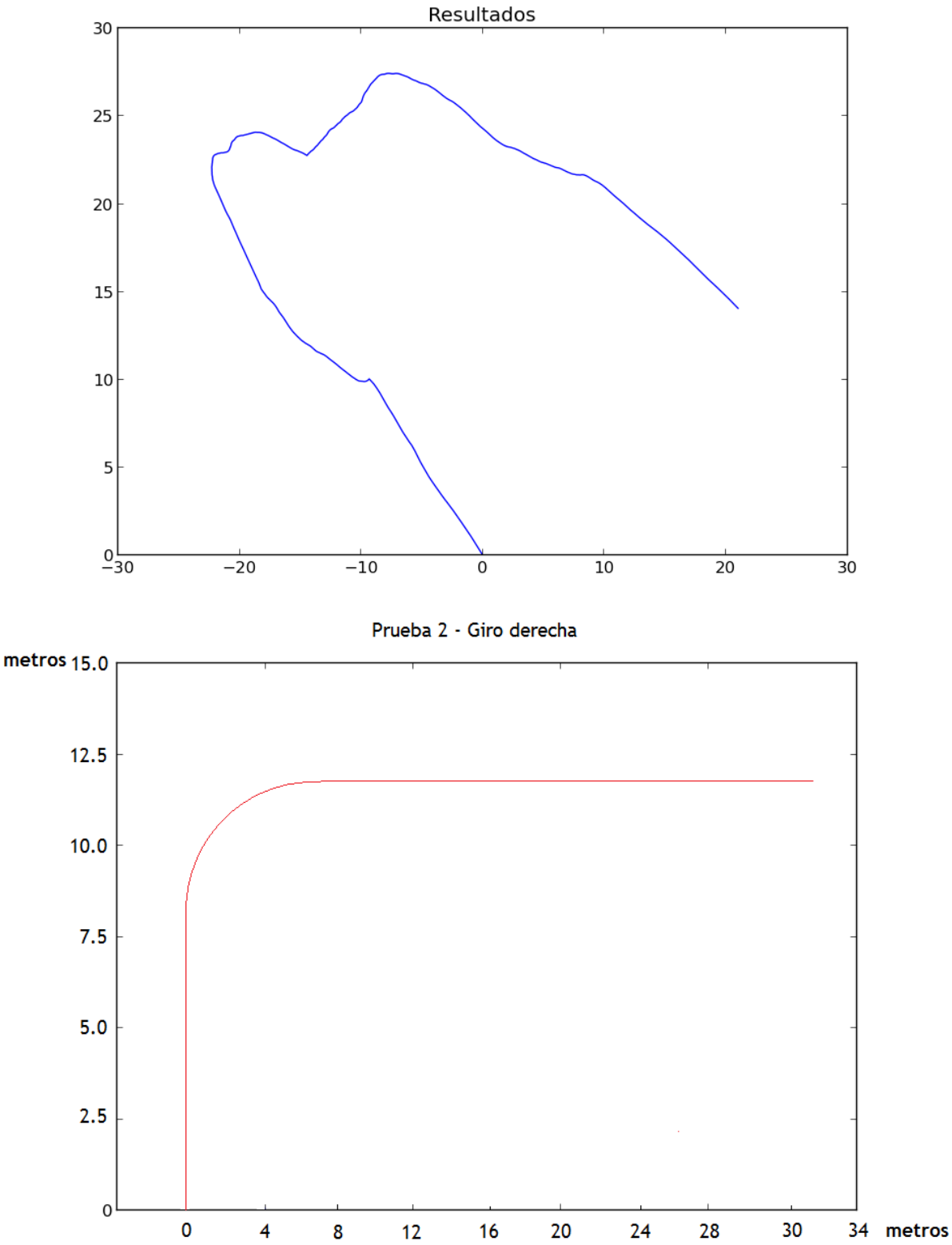

Figura 13: Graficas resultados, sistema (azul), ground truth (rojo).

La tabla 2 siguiente muestra la diferencia entre la estimación del sistema de odometría visual diseñado y el Ground Truth (G.T.) para ciertos fotogramas de la secuencia de imágenes de esta prueba.

Instante	X (G.T.)	Y (G.T.)	X (estimación)	Y (estimación)	Error (D)
0	0,0	0,0	0,0	0,0	0
1	0,127	2,358	-10,636	10,023	13,2134172
2	0,114	6,035	-19,176	20,145	23,8997113
3	-0,105	11,297	-23,087	11,521	22,9830916
4	8,209	12,325	1,982	25,493	14,5661166
5	17,278	12,466	9,101	23,861	14,0253112
6	27,102	12,352	18,762	17,238	9,66584688
Final	32,226	12,474	21,218	15,362	11,3805364

Tabla 2: Estimación del sistema de odometría visual diseñado y el Ground Truth para giro derecha.

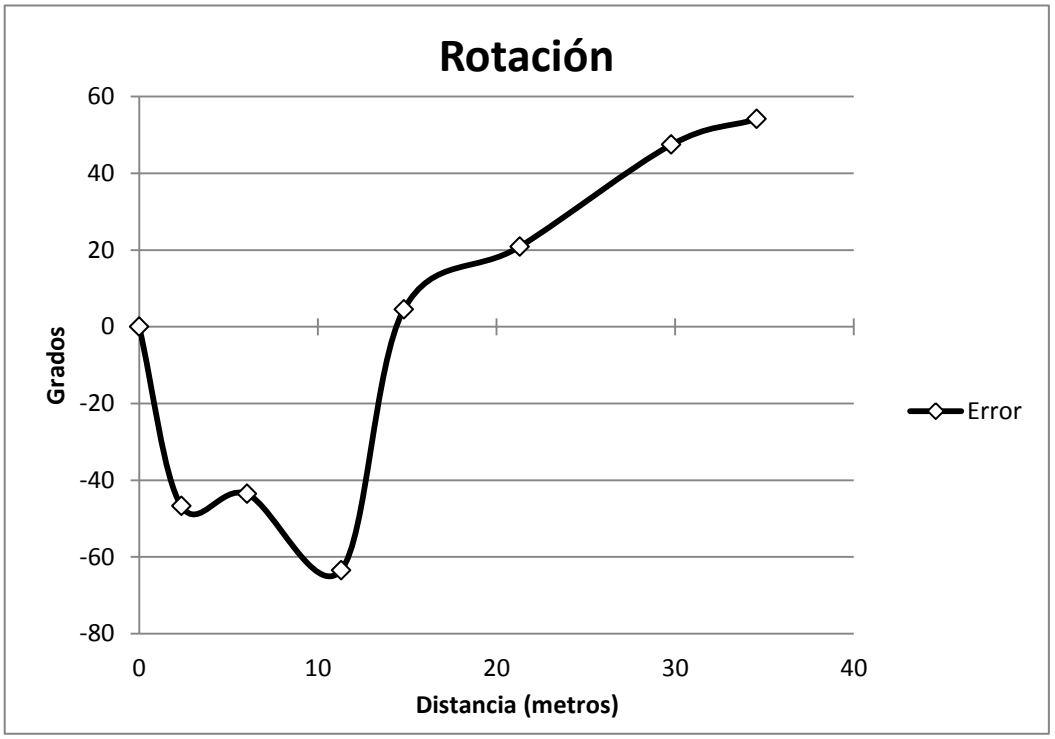

Figura 14: Error de rotación en función de la trayectoria. Prueba giro derecha.

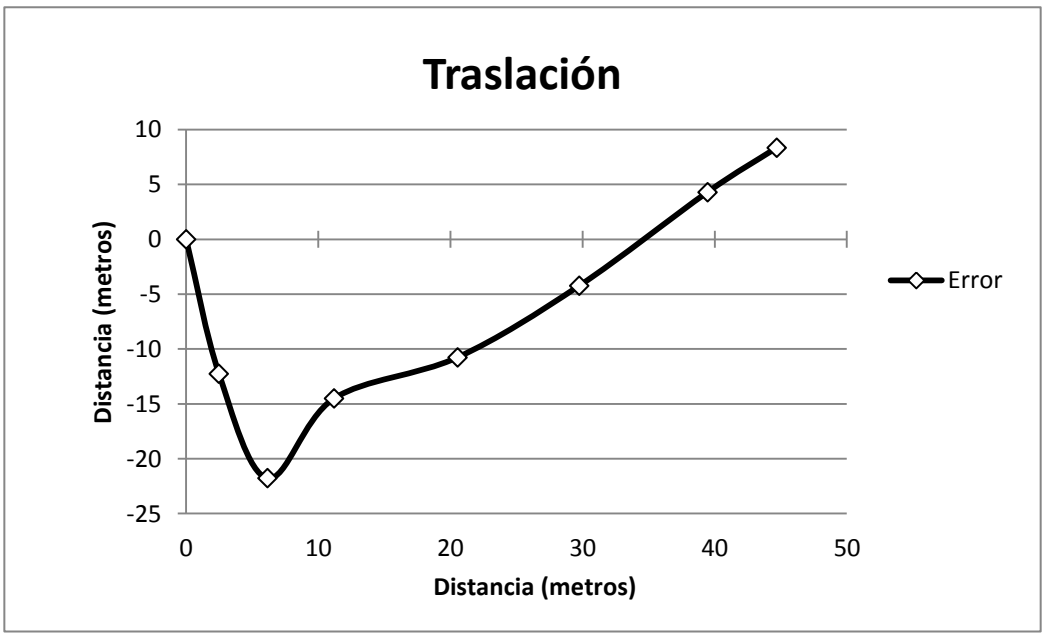

Figura 15: Error de traslación en función de la trayectoria. Prueba giro derecha.

4.4 Prueba avance y retroceso.

Esta secuencia fue capturada en un ambiente de carretera y se compone de 1246 fotogramas que representan 41 segundos de recorrido (410 fotogramas después de editado) con un tiempo de ejecución de 423 segundos. Para esta prueba es importante mencionar que el vehículo se desplaza a una velocidad prácticamente constante de unos 20 Km/h.

En la figura 16 se puede observar la trayectoria del vehículo en una visión aérea extraída de google Maps.

Figura 16: Plano de los exteriores donde se ha realizado la tercera prueba de trayectoria de avance y retroceso.

La grafica obtenida de la secuencia de fotogramas extraída del video refleja el resultado para esta prueba:

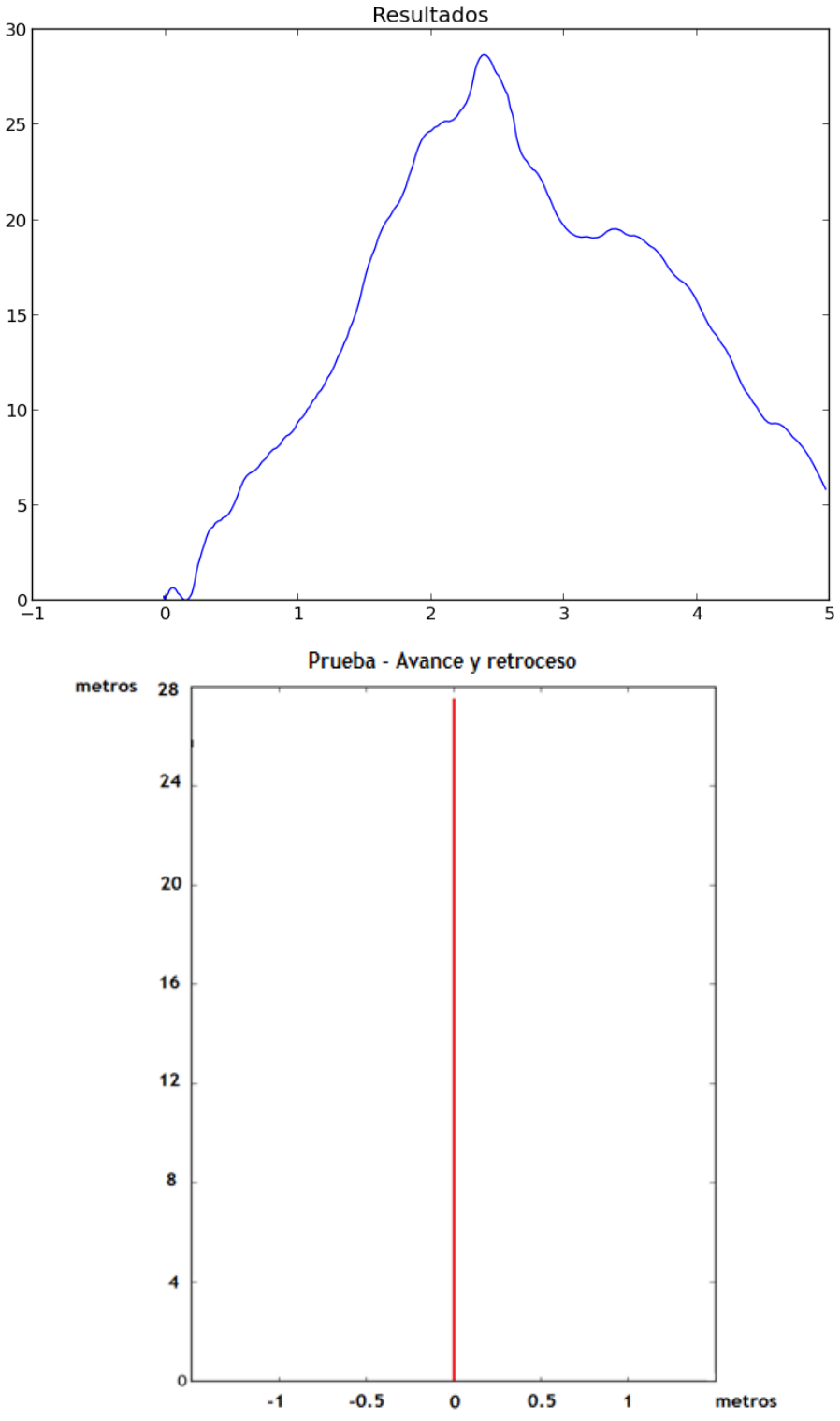

Figura 17: Graficas resultados, sistema (azul), ground truth (rojo).

La tabla 3 siguiente muestra la diferencia entre la estimación del sistema de odometría visual diseñado y el Ground Truth (G.T.) para ciertos fotogramas de la secuencia de imágenes de esta prueba.

Instante	X (G.T.)	Y (G.T.)	X (estimación)	Y (estimación)	Error (D)
0	0,0	0.0	0	0	0
1	0,0	8.248	1,345	9,653	1,94500643
2	0,0	16.209	1,823	18,259	2,74332444
3	0,0	24.455	2,129	24,966	2,18946614
4	0,0	25.374	3,070	19,369	6,74425126
5	0,0	16.943	3,855	17,767	3,9420808
6	0,0	8.119	4,531	9,106	4,63725458
Final	0,0	0,0	4,991	5,967	7,7791497

Tabla 3: Estimación del sistema de odometría visual diseñado y el Ground Truth para avance y retroceso.

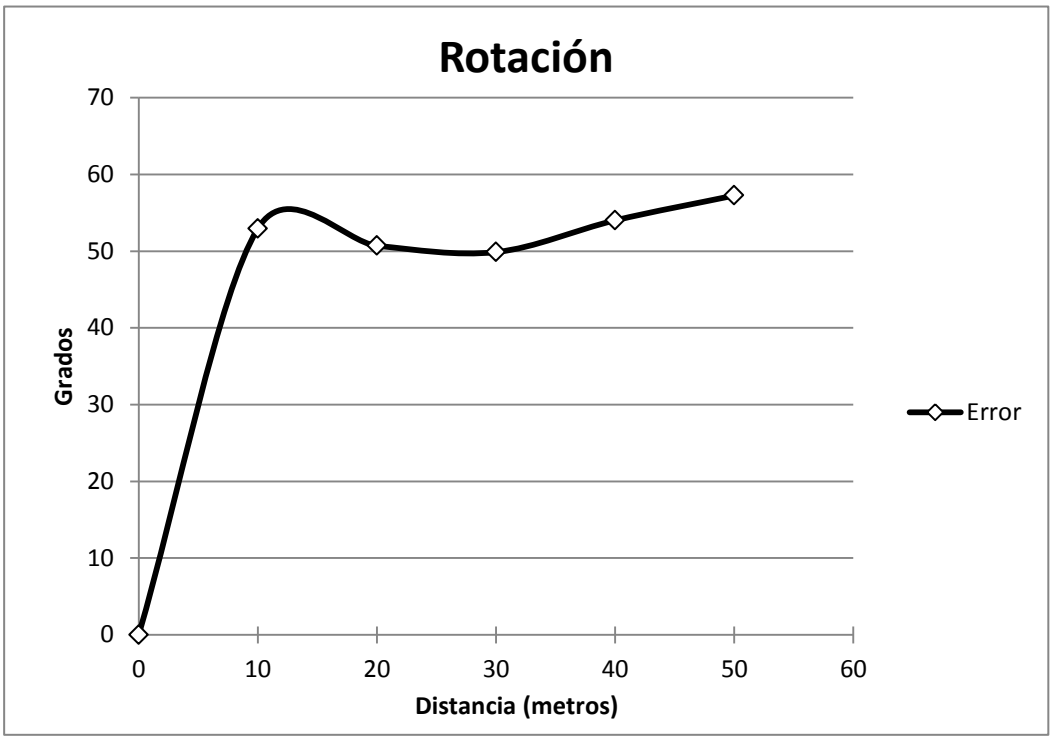

Figura 18: Error de rotación en función de la trayectoria. Prueba avance y retroceso.

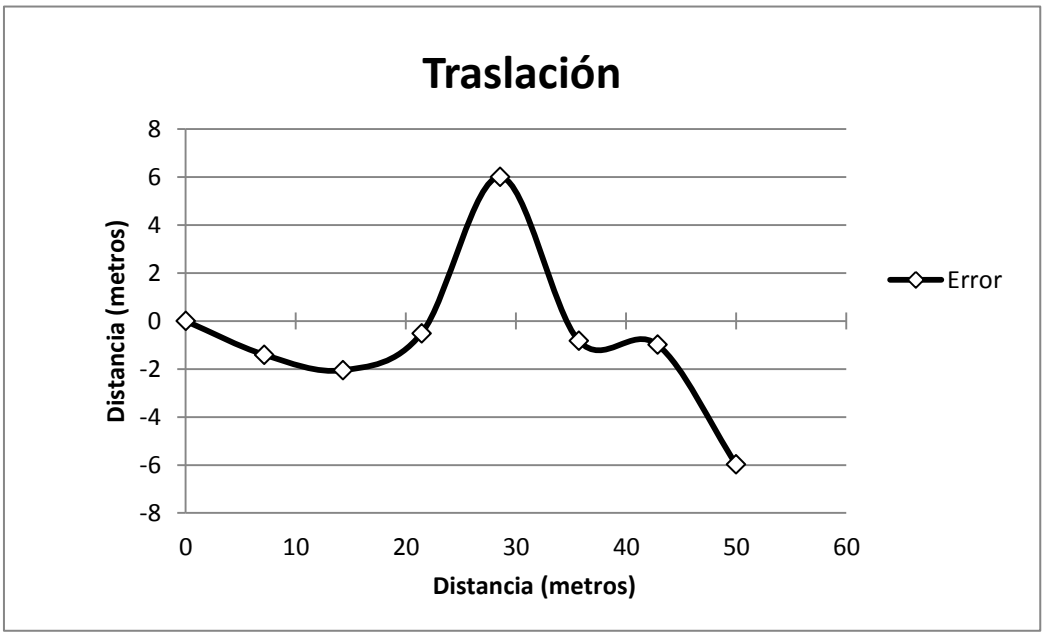

Figura 19: Error de traslación en función de la trayectoria. Prueba avance y retroceso.

Capítulo 5

Conclusiones y trabajo futuro

5.1 Conclusiones

En este proyecto se desarrolló un sistema de odometría visual cuyo funcionamiento se puede resumir de la siguiente manera:

- Utiliza una cámara digital común para la captación de imágenes.
- Utiliza extractor de características de tipo blobs. (SURF).
- Implementa un emparejamiento de características por detección del vecino más cercano y emparejamiento restringido. (FLANN).
- Obtiene matriz esencial a través de la matriz de calibración de la cámara y la matriz fundamental.
- Extrae la matriz de rotación y el vector de traslación a partir de la matriz esencial.

Los resultados obtenidos confirman que no es un buen sistema de visión por computador. El desarrollo de este proyecto está basado únicamente en módulos de la librería OpenCV, llegando a la conclusión que varios de los cálculos realizados se pueden mejorar llevando a cabo una implementación más compleja.

El lenguaje seleccionado para el desarrollo es Python, un lenguaje interpretado que merma bastante la velocidad de cómputo en un sistema de estas características.

La captura de imágenes que aun siendo de alta calidad no son ni mucho menos recomendables para aplicar en un sistema de visión por computador que se desee utilizar en aplicaciones reales.

Otro problema evidente es el factor de escala aplicado a la traslación, siendo en este caso introducido de forma manual dependiendo de las características de la secuencia de imágenes. Este es un factor fundamental en el desarrollo de sistemas de localización basados en visión por computador.

El error que se produce en el cálculo de la trayectoria se va acumulando durante a medida que se van procesando imágenes, algunos autores proponen corregir este error a medida que se van obteniendo nuevas imágenes, algo que no se ha tenido en cuenta en este proyecto.

Por último y teniendo en cuenta el tiempo de ejecución del algoritmo, el sistema desarrollado es inviable para su uso en tiempo real.

5.2 Trabajo Futuro

Para mejorar el presente sistema de visión por computador se pueden realizar diversos proyectos futuros, a continuación se proponen algunas de las posibles mejoras:

- Para mejorar considerablemente el proyecto se podría cambiar el lenguaje de programación usado, implementando los módulos de OpenCV con el lenguaje de programación C++ que siendo compatible en su totalidad con la librería, es considerablemente más rápido en la ejecución de este tipo de algoritmos.
- Para la captura de imágenes se podría implementar una cámara de visión estereoscópica de mejores prestaciones.
- Implementar un método para la determinación de la escala absoluta del movimiento de tal manera que se obtenga una trayectoria real del movimiento del vehículo.
- Mejorar la obtención de la matriz de rotación y el vector de traslación con el fin de corregir el error acumulativo en la extracción del movimiento.
- Investigar la implementación de una estrategia de extracción de características con la que se obtenga mejores resultados.
- Implementación de un sistema de SLAM (Simultaneous Localization and Mapping) que permita tener un modelo del ambiente que rodea al vehículo y que permita identificar cuando el vehículo pasa por un lugar por el que ha pasado con anterioridad (loop closure, V SLAM).
- En el enfoque propuesto en este proyecto se estima el movimiento únicamente a partir de 2 fotogramas. Un trabajo futuro podría intentar involucrar un número mayor de fotogramas buscando mejorar la consistencia global de la estimación. En este sentido podrían tenerse en cuenta técnicas como local bundle adjustment, localización basada en grafos, localización de Markov, filtros de Kalman etc.

Capítulo 6

Conclusions and future work

6.1 Conclusions

In this project visual odometry system whose operation can be summarized as follows developed:

- Use a common digital camera for capturing images.
- Use feature extractor type BLOB. (SURF).
- Implement a match detection feature nearest neighbor and restricted pairing. (FLANN).
- Gets essential matrix through the matrix camera calibration and the fundamental matrix.
- Extract the rotation matrix and the translation vector from the essential matrix.

The results confirm that it is not a good system of computer vision. The development of this project is based solely on the OpenCV library modules, concluding that several of the calculations can be improved by carrying out a more complex implementation.

The language selected for development is Python, an interpreted language that undermines enough computing speed of a system of this kind.
Capturing images while of high quality are far less desirable to apply a computer vision system you want to use in real applications.
Another obvious problem is the scale factor applied to translation, in this case being entered manually depending on the charac-teristics of the image sequence. This is a fundamental factor in the development of tracking systems based on computer vision.

The error occurs in the calculation of the trajectory is accumulated during as images are being processed, some authors propose to correct this error as new images are obtained, which has not been considered in this project.

Finally, taking into account the running time of the algorithm, the developed system is unfeasible for use in real time.

6.2 Future work.

To improve the present system of computer vision can make various future projects then proposed some possible improvements:

- To improve considerably the project could change the programming language used by implementing modules OpenCV with C ++ programming language that is compatible with the entire library, is considerably faster in the execution of such algorithms.
- To capture images could implement a stereoscopic camera for better performance.
- Implement a method for determining the absolute scale of motion so that an actual vehicle path of movement is obtained.
- Improving obtaining the rotation matrix and the translation vector in order to correct the cumulative error in the extraction movement.
- Investigate the implementation of a strategy of feature extraction with the best results will be obtained.
- Implementation of a system of SLAM (Simultaneous Localization and Mapping) that allows to have a model of the environment surrounding the vehicle and to identify when the vehicle passes through a place that has happened before (loop closure, V SLAM).
- The approach proposed in this project is estimated movement only from 2 frames. Future work could try to involve a greater number of frames seeking to improve the overall consistency of the estimate. In this regard they could be taken into account technical and local bundle adjustment, based on graphs location, location Markov Kalman filters etc.

Capítulo 7

Costes del proyecto

En este capítulo se presenta una relación aproximada de los costes en los que se ha incurrido durante la elaboración del proyecto. Se incluyen tanto costes de personal como de materiales.

7.1 Ejecución material.

7.1.1 Materiales

Para el cálculo del coste de los materiales, que se pueden reutilizar de forma sencilla en otras tareas, se ha calculado la repercusión de la amortización lineal de su coste de adquisición en el tiempo utilizado para el presente proyecto. Es por ello que se ha indicado en la tabla para dichos materiales: valor de adquisición / periodo máximo de amortización. Los medios que se utilizarán para el desarrollo del proyecto serán:

7.1.1.1 Hardware y software.

Descripción	Cant.	Precio unitario	Total
Ordenador (ASUS ROG G20AJ)	4 meses	1400€/24 meses	233.35€
Cámara SJCAM (Sj4000)	4 meses	115€	115€
Impresora color	4 meses	150€/12 meses	50€
Bibliotecas (OpenCV, matplotlib,...)	4 meses	0€	0€
Framework SublimeText	4 meses	70€/12 meses	23.35
Sistema Operativo Linux UBUNTU 12.40	4 meses	0€	0€
Software OpenOffice	4 meses	0€	0€
Software Gimp (Tratamiento de imágenes)	4 meses	0€	0€
TOTAL			**411.70 €**

Tabla 4: Software y Hardware utilizado en la elaboración del proyecto.

7.1.1.2 Material fungible

Descripción	Cant.	Precio	TIEMPO
Papel y material oficina	1	20€	20€
Tóner impresora	1	15€	15€
Otro material (cables, soportes, etc.)	1	150€	150€
Mecanizado (soporte cámara)	1	15€	15€
TOTAL			**200.00€**

Tabla 5: Material fungible utilizado en la elaboración del proyecto.

7.1.1.3 Total coste de materiales

Total coste de materiales = 411.70 € + 200.00 € = **611.70 €**

7.1.2 Mano de Obra

ETAPA	TIEMPO
Estudio y comprensión del problema a resolver	50 h
Familiarización con las herramientas (Python, OpenCV,...)	30 h
Implementación del código	250 h
Pruebas y corrección de errores	80 h
Ajuste de parámetros y mejoras	100 h
Obtención de resultados	20 h
Memoria del proyecto	100 h
TOTAL	**680 h**

Tabla 6: Tiempo estimado para cada fase del proyecto

Estimando 20 días hábiles al mes, el tiempo de desarrollo seria de: 680 horas / 8 horas/día / 20 días hábiles = 4 meses. Y considerando el coste del ingeniero diario en 240 €, resulta:

Total coste mano de obra: 4 meses x 20 días x 240€/día = 19.200,00€

7.2 Gastos generales

Se trata de estimar el coste del uso de instalaciones y otros servicios relacionados con el proyecto.

Para ello estimaremos el coste de distintos servicios:

Descripción	Cant.	Precio	TIEMPO
Energía	4 meses	75 € / mes	300 €
Telecomunicaciones	4 meses	50 € / mes	100 €
TOTAL			**400.00 €**

Tabla 7: Gastos generales

Total coste de gastos generales = 400,00€

7.3 Coste total de ejecución

Descripción	TIEMPO
Total coste materiales	**611,70 €**
Total costes mano de obra	19.200,00 €
Total costes generales	400,00 €
TOTAL	**20.211,70 €**

Tabla 8: Costes de ejecución.

7.4 Presupuesto total

Descripción	TIEMPO
Costes de ejecución	20.211,70 **€**
I.G.I.C. (7%)	1.414,81 €
TOTAL	**21.616,51 €**

Tabla 9: Presupuesto total.

El importe total de este proyecto asciende a la cantidad de veintiún mil seiscientos dieciséis euros con cincuenta y un céntimos de euro.

Bibliografía

1. C. Harris and M. Stephens. “A combined corner and edge detector”. Proc. Fourth Alvey Vision Conference.
2. C. Schmid, R. Mohr and C. Bauckhage. “ Evaluation of Interest Point Detectors”. International Journal of Computer Vision.
3. G. Xu and Z. Zhang. “Epipolar Geometry in Stereo, Motion, and Object Recognition: A Unified Approach”. Kluwer Academic Publishers Norwell, MA, USA. 1996
4. M. A. Fischler and R. C. Bolles. “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography”. Communications of the ACM.
5. R. Hartley and A. Zisserman. “Multiple View Geometry in Computer Vision”. Cambridge University Press, 2004.
6. Open Computer Vision Library https://opencv-python-tutroals.readthedocs.org/en/latest/
7. Técnicas de visión. http://tecnicasdevision.blogspot.com.es/2014_05_01_archive.html.
8. Z. Zhang and O. D. Faugeras. “Estimation of displacements from two 3-D frames obtained from stereo”. IEEE Transactions on Pattern Analysis and Machine Intelligence.
9. D. Nister, O. Narodistsky and J. Bergen. “Visual Odometry”. Proc. IEEE Conference on Computer Vision and Pattern Recognition.
10. M. Agrawal and K. Konolige.“Real-time Localization in Outdoor Environents using Stereo Vision and Inexpensive GPS”. 18th International Conference on Pattern Recognition.
11. Y. Cheng, M. W. Maimone, L. Matthies. “Visual Odometry on the Mars Exploration Rovers”. IEEE Robotics and Automation Magazine.
13. D. SCARAMUZZA y F. FRAUNDORFER. ((Visual Odometry [Tutorial])). En: Robotics Automation Magazine, IEEE 18.4 (Dec.).
14. D. SCARAMUZZA, F. FRAUNDORFER y R. SIEGWART. (Real-time monocular visual odometry for on-road vehicles with 1-point RANSAC). Robotics and Automation, ICRA '09. IEEE International Conference on. 2009.
15. Google Maps 2015. http://maps.google.com.
16. Wikipedia.