



*ugr* | Universidad  
de **Granada**

TRABAJO FIN DE GRADO  
GRADO EN INGENIERÍA INFORMÁTICA

# Técnicas de estimación de geometría 3D usando retinas artificiales

---

## Autor

José Antonio Díaz Ramírez

## Directores

Francisco Barranco Expósito



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

—  
Granada, Septiembre de 2021



# Técnicas de estimación de geometría 3D usando retinas artificiales

José Antonio Díaz Ramírez

**Palabras clave:** reconstrucción 3D, eventos,sensores basados en eventos, simulador de eventos, estimador posición de la cámara, Robot Operating System, visión por computador.

## Resumen

En el presente Trabajo Fin de Grado se ha desarrollado un sistema de reconstrucción 3D con la información captada por un sensor de eventos.

Este proyecto se centra en el estudio y la integración del sistema EMVS capaz de reconstruir un entorno 3D a través de eventos, con la capacidad de obtenerlos desde un vídeo tradicional gracias al simulador de eventos V2E y la ayuda de SVO para la estimación del movimiento de la cámara en la escena.

El trabajo se ha implementado sobre el marco de trabajo ROS (Robot Operating System), un middleware que permite la integración de diferentes componentes de forma transparente en un mismo sistema distribuido. Para alcanzar el correcto funcionamiento del proyecto se ha recurrido a la integración de módulos propios, lo cual ha facilitado la reconstrucción 3D a partir de un vídeo grabado con una cámara tradicional.

Esta memoria muestra además el estudio seguido durante el proyecto acerca de la información captada por sensores basados en eventos, las diferentes técnicas tradicionales y adaptadas a eventos para la reconstrucción 3D, para a continuación mostrar el diseño del sistema al completo, exponer los diferentes experimentos llevados a cabo, y finalmente, discutir los resultados obtenidos entre el conjunto de datos y vídeos grabados por mi mismo.

# 3D geometry estimation techniques using artificial retinas

José Antonio Díaz Ramírez

**Keywords:** 3D reconstruction, events, event-based sensor, event simulator, motion sensor, Robot Operating System, computer vision.

## Abstract

The aim of this project has been to develop a 3D reconstruction system with the information captured by an event-based camera.

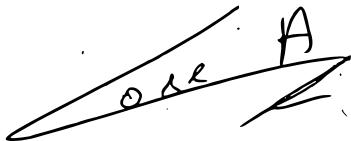
This project focuses on the study and integration of the EMVS system, which is capable of making a 3D environment reconstruction through events. This system can obtain these events from a traditional video thanks to the V2E event simulator and also to SVO for the estimation of camera motion in the scene.

This work has been implemented on the ROS (Robot Operating System) framework, a middleware that allows the integration of different components in a transparent way in the same distributed system. In order to achieve the correct functioning of the project, self-made modules have been integrated, which has facilitated the 3D reconstruction from a video recorded with a traditional camera.

This report also shows the route followed throughout the project with regards to the information captured by event-based sensors, the different traditional and event-driven techniques for 3D reconstruction, to then show the complete system design, present the different experiments which have been carried out, and finally, discuss the results obtained from the dataset and videos shot by myself.

---

Yo, **José Antonio Díaz Ramírez**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77556334N, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

A handwritten signature in black ink. The name "José Antonio" is written in cursive script, followed by "Díaz Ramírez" in a slightly more formal hand. A small letter "A" is written above the signature.

Fdo: José Antonio Díaz Ramírez

Granada a 6 de Septiembre de 2021 .

---

**D. Francisco Barranco Expósito**, Profesor del Área de Arquitectura y Tecnología de Computadores del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

**Informan:**

Que el presente trabajo, titulado *Técnicas de estimación de geometría 3D usando retinas artificiales*, ha sido realizado bajo su supervisión por **José Antonio Díaz Ramírez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 9 de Septiembre de 2021.

**Los directores:**

**Francisco Barranco Expósito**



### Declaración de Originalidad del TFG

D./Dña. José Antonio Díaz Ramírez, con DNI  
(NIE o pasaporte) 77556334N, declaro que el presente Trabajo de Fin de Grado es original, no habiéndose utilizado fuente sin ser citadas debidamente. De no cumplir con este compromiso, soy consciente de que, de acuerdo con la [Normativa de Evaluación y de Calificación de los estudiantes de la Universidad de Granada](#) de 20 de mayo de 2013, esto *conllevará automáticamente la calificación numérica de cero [...] independientemente del resto de las calificaciones que el estudiante hubiera obtenido. Esta consecuencia debe entenderse sin perjuicio de las responsabilidades disciplinarias en las que pudieran incurrir los estudiantes que plagie.*

Para que conste así lo firmo el 6 de Septiembre de 2021

Firma del alumno

A handwritten signature in black ink, consisting of a stylized "J" followed by "osé", "Antonio", "Díaz", and "Ramírez". Above the signature, there is a small letter "A".

José Antonio Díaz Ramírez

# Agradecimientos

Me gustaría agradecer primero a mi familia durante estos 4 años de carrera universitaria, dándome su total apoyo y comprensión hacia mí en los distintos retos que han ido apareciendo a lo largo de este camino.

También a todos mis amigos, tanto a los de toda la vida como a los nuevos que me ha dado esta etapa universitaria, que me han ayudado tanto en mi vida personal como académica.

Agradecer a mi tutor Francisco Barranco por ser mi guía durante el proyecto y ayudarme siempre que lo necesitaba.

Por último, quiero agradecer a todos aquellos profesores, que además de conocimiento, han conseguido transmitirme el interés por sus asignaturas, haciéndome más fácil la elección del camino a seguir una vez terminado el grado universitario.

# Índice general

<b>1. Introducción</b>	<b>20</b>
1.1. Motivación . . . . .	21
1.2. Estado del arte . . . . .	22
1.2.1. Métodos usados para la conducción autónoma de drones o SLAM en entornos dinámicos . . . . .	22
1.2.2. Cámara de eventos . . . . .	24
1.2.3. Simuladores de eventos . . . . .	28
1.2.4. Estimación de la posición de la cámara . . . . .	31
1.2.5. Algoritmos para la reconstrucción 3D y estimación de profundidad a través de eventos . . . . .	38
1.3. Objetivos . . . . .	42
1.4. Estructura del proyecto . . . . .	43
<b>2. Gestión del proyecto</b>	<b>45</b>
2.1. Metodología . . . . .	45
2.2. Planificación temporal . . . . .	46
2.3. Hitos . . . . .	48
2.4. Presupuesto . . . . .	49
2.4.1. Recursos Hardware . . . . .	50
2.4.2. Recursos Software . . . . .	50
2.4.3. Recursos Humanos . . . . .	50
<b>3. Entorno Operacional</b>	<b>52</b>
3.1. ROS . . . . .	52
3.2. Ordenador propio y SO (Ubuntu 20.04) . . . . .	58
3.3. Servidor UGR con tarjeta gráfica Nvidia RTX 2080 TI . . . . .	59
3.4. iPhone SE (Cámara) . . . . .	59
<b>4. Diseño</b>	<b>60</b>
4.1. Casos de uso . . . . .	62
4.2. Requisitos . . . . .	64
4.2.1. Funcionales . . . . .	64
4.2.2. No funcionales . . . . .	65

4.3. Diagrama de Flujo . . . . .	67
<b>5. Implementación</b>	<b>69</b>
5.1. V2E . . . . .	69
5.2. SVO . . . . .	70
5.3. EMVS . . . . .	72
5.4. Crear_rosbag . . . . .	72
5.5. Manual de usuario . . . . .	82
5.5.1. V2E . . . . .	82
5.5.2. Video2Bag . . . . .	82
5.5.3. SVO . . . . .	83
5.5.4. Crear_rosbag . . . . .	84
5.5.5. EMVS . . . . .	85
<b>6. Problemas durante el desarrollo</b>	<b>86</b>
6.1. Restricciones Hardware . . . . .	86
6.2. Restricciones Software . . . . .	86
6.3. Restricciones sanitarias por el Covid-19 . . . . .	86
6.4. Problemas técnicos . . . . .	87
6.4.1. Compatibilidad de Anaconda con ROS Noetic/Melodic . . . . .	87
6.4.2. Limitaciones generadas por V2E . . . . .	87
6.4.3. Servidor compartido . . . . .	88
6.4.4. Modificaciones de código para correcto funcionamiento en V2E, SVO y EMVS . . . . .	88
6.4.5. Imprevistos en la experimentación (SVO y EMVS) . . . . .	92
<b>7. Experimentos</b>	<b>93</b>
7.1. SVO . . . . .	93
7.1.1. Prueba con vídeo del Dataset Zurich (oficina) . . . . .	94
7.1.2. Pruebas en interiores con un tablero de ajedrez con parámetros por defecto . . . . .	95
7.1.3. Pruebas en interiores con un tablero de ajedrez y con cambio de parámetros . . . . .	97
7.1.4. Pruebas en interiores con estampados extravagantes y con cambio de parámetros . . . . .	99
7.1.5. Pruebas en interiores con texturas extravagantes, resolución 720p y con cambio de parámetros . . . . .	102
7.1.6. Pruebas en interiores con Arucos/Graffiti y con cambio de parámetros . . . . .	107
7.1.7. Discusión de resultados . . . . .	110
7.2. V2E . . . . .	112
7.2.1. Tamaño . . . . .	112
7.2.2. Tiempos de ejecución . . . . .	118
7.3. Reconstrucción 3D . . . . .	122

---

7.3.1. Ejemplos propios . . . . .	123
7.3.2. Ejemplos Dataset Zurich . . . . .	135
7.3.3. Discusión resultados Zurich - Propios . . . . .	141
<b>8. Conclusiones y trabajo futuro</b> . . . . .	<b>143</b>
8.1. Conclusiones . . . . .	143
8.2. Análisis de objetivos . . . . .	145
8.3. Trabajo futuro . . . . .	146
<b>Bibliografia</b>	<b>151</b>

# Índice de figuras

1.1.	SLAM basado en un agente. Izquierda: Agente con el sistema equipado. Centro: Imagen de Google Earth del edificio Derecha: Mapeado 3D sin conexión del entorno del edificio tras moverse el agente por el recorrido en rojo. . . . .	24
1.2.	Comparativa entre cámaras tradicionales y sensores DVS, captura de la animación que podemos encontrar en <a href="https://youtube.com/watch?v=LauQ6LWTkxM">https://youtube.com/watch?v=LauQ6LWTkxM</a> . . . . .	25
1.3.	Pasos del v2e para simular eventos como los sensores DVS . . . . .	29
1.4.	Ejemplo de la posición y movimiento de la cámara extraída . . . . .	32
1.5.	Imagen conseguida con el sensor . . . . .	34
1.6.	Proyección del mapa 3D semidenso . . . . .	34
1.7.	Ejemplo extracción características robustas . . . . .	36
1.8.	Cambiando la posición relativa Tk,k-1 entre el fotograma anterior (k-1) y el nuevo (k), mueve implícitamente la proyección de puntos en el espacio para el nuevo fotograma. El modelo disperso para la alineación de la imagen pretende minimizar el error cometido en los cuadrados azules entre fotogramas que pertenezcan a los mismos puntos del espacio . . . . .	37
1.9.	Rayos propagados con MVS usando fotogramas clásicos, dependiendo de la velocidad de la cámara. . . . .	39
1.10.	Rayos propagados con eventos, disparados cada vez que se mueve el sensor . . . . .	39
1.11.	Creación del DS1 contando los rayos que pasan por campo proyectado desde el punto RV. . . . .	40
1.12.	a) Imagen del entorno virtual b) DS1 en profundidad corta c) DS1 en profundidad media d) DS1 en profundidad larga. . . . .	41
1.13.	DSI con la densidad de rayos. . . . .	41
1.14.	Mapa semidenso de profundidad. . . . .	41
1.15.	Nube de puntos. . . . .	42
2.1.	Esquema del desarrollo de prototipos . . . . .	45
2.2.	Diagrama de Gantt . . . . .	46

3.1. Estructura del mensaje CameraInfo . . . . .	54
3.2. Estructura del mensaje EventArray . . . . .	55
3.3. Estructura del mensaje PoseStamped . . . . .	56
4.1. Diseño del sistema al completo . . . . .	60
4.2. Diagrama de Flujo . . . . .	67
5.1. Salida de la ejecución del simulador V2E . . . . .	69
5.2. Fotograma en escala de grises del video original generado por V2E . . . . .	70
5.3. Fotograma RGB del vídeo grabado por la cámara del iPhone	70
5.4. Fotograma con eventos positivos (blanco) y negativos (negro) del vídeo DVS generado por V2E . . . . .	70
5.5. Archivo YAML con información sobre la cámara . . . . .	71
5.6. Script video2bag . . . . .	72
5.7. Diagrama flujo de Crear_rosbag . . . . .	74
5.8. CMakeList.txt del paquete crear_rosbag . . . . .	74
5.9. package.xml del paquete crear_rosbag . . . . .	75
5.10. Fichero launch del paquete crear_rosbag . . . . .	75
5.11. Librerías necesarias para funcionamiento del paquete crear_rosbag . . . . .	75
5.12. Comienzo del __main__ de crear_rosbag.py . . . . .	76
5.13. Comienzo función parse_dataset de crear_rosbag.py . . . . .	76
5.14. Asignación de datos de la cámara para su salida en la función parse_dataset . . . . .	76
5.15. Tratamiento del archivo con las posiciones guardadas de la función parse_dataset . . . . .	77
5.16. Ejemplo de salida de una posición de la cámara publicada por el topic /svo/pose . . . . .	78
5.17. Función make_camera_msg . . . . .	78
5.18. Función make_pose_msg . . . . .	79
5.19. Bucle recorriendo cada una de las marcas de tiempo . . . . .	80
5.20. Función parse_events . . . . .	80
5.21. Comienzo archivo v2e-dvs-events.txt . . . . .	81
5.22. Función make_events . . . . .	81
5.23. Ejemplo parámetros configuración EMVS . . . . .	81
5.24. Nombre fichero calibración de la cámara . . . . .	84
5.25. Nombre fichero posiciones sacadas por SVO . . . . .	84
5.26. Nombre fichero con el conjunto de eventos . . . . .	84
5.27. Nombre del fichero binario de salida con formato bag . . . . .	85
6.1. Líneas a comentar en el constructor del archivo emulator.py .	89
6.2. Líneas a comentar en la función cleanup del archivo emulator.py	89

6.3. Líneas a comentar en la función generate_events del archivo emulator.py . . . . .	89
6.4. Línea 267 a cambiar del archivo sparse_img_align.cpp . . . . .	90
6.5. Línea 57 del archivo test_feature_detection.cpp . . . . .	90
6.6. Línea 141 a cambiar del archivo visualizer.cpp . . . . .	90
6.7. Línea 149 a cambiar del archivo visualizer.cpp . . . . .	90
6.8. Línea 112 del archivo pinhole_camera.cpp . . . . .	90
6.9. Línea 48 del archivo homography.cpp . . . . .	90
6.10. Línea 237 del archivo img_align.cpp . . . . .	90
6.11. Línea 437 del archivo img_align.cpp . . . . .	91
6.12. Línea a comentar en el fichero main.cpp del EMVS . . . . .	91
6.13. Línea 8 del fichero CMakeList.txt del mapper_emvs . . . . .	91
7.1. Comienzo de la estimación de un vídeo con SVO . . . . .	94
7.2. Seguimiento de la estimación de un vídeo con SVO . . . . .	94
7.3. Trayectoria resultante de la estimación de un vídeo con SVO . . . . .	94
7.4. Comienzo de prueba Dynamic en el SVO . . . . .	95
7.5. Seguimiento de prueba Dynamic en el SVO . . . . .	95
7.6. Seguimiento de características para inicializar mapa de triangulación de la prueba con tablero encima de la cama . . . . .	96
7.7. Primer fotograma con extracción de características de la prueba con tablero encima de la cama . . . . .	96
7.8. Salida terminal triangulación del mapa de la prueba con tablero encima de la cama . . . . .	96
7.9. Extracción de características durante el vídeo de la prueba con tablero encima de la cama . . . . .	97
7.10. Salida terminal durante el vídeo de la prueba con tablero encima de la cama . . . . .	97
7.11. Seguimiento de características para inicializar mapa de triangulación en prueba en interior con tablero de ajedrez con nuevos parámetros . . . . .	98
7.12. Primer fotograma con extracción de características en prueba en interior con tablero de ajedrez con nuevos parámetros . . . . .	98
7.13. Salida terminal mapa triangulación en prueba en interior con tablero de ajedrez . . . . .	98
7.14. Extracción de características durante el vídeo en prueba en interior con tablero de ajedrez . . . . .	98
7.15. Seguimiento de características robustas para triangulación en prueba escritorio con tablero, ordenador y monitor . . . . .	99
7.16. Primer fotograma con extracción de características en prueba escritorio con tablero, ordenador y monitor . . . . .	99
7.17. Salida terminal con el mapa de triangulación en prueba escritorio con tablero, ordenador y monitor . . . . .	99

7.18. Extracción de características durante el vídeo en prueba escritorio con tablero, ordenador y monitor . . . . .	100
7.19. Extracción de características para relocalización de la escena en prueba escritorio con tablero, ordenador y monitor . . . . .	100
7.20. Salida terminal con mensaje de relocalización en prueba escritorio con tablero, ordenador y monitor . . . . .	100
7.21. Seguimiento de características robustas para triangulación en prueba escritorio con muñeco . . . . .	101
7.22. Primer fotograma con extracción de características en prueba escritorio con muñeco . . . . .	101
7.23. Salida terminal con el mapa de triangulación en prueba escritorio con muñeco . . . . .	101
7.24. Inicio de estimación en prueba escritorio con muñeco . . . . .	101
7.25. Estimación en fotogramas finales en la prueba escritorio con muñeco . . . . .	101
7.26. Extracción de características durante el vídeo en prueba escritorio con muñeco . . . . .	102
7.27. Seguimiento de características robustas para triangulación en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles . . . . .	102
7.28. Primer fotograma con extracción de características en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles . . . . .	102
7.29. Salida terminal con el mapa de triangulación en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles . . . . .	103
7.30. Extracción de características durante el vídeo en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles . . . . .	103
7.31. Extracción de características posterior en el vídeo de prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles . . . . .	103
7.32. Inicio de estimación en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles . . . . .	104
7.33. Estimación final en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles . . . . .	104
7.34. Salida terminal con mensaje de relocalización en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles . . . . .	104
7.35. Búsqueda de nuevos puntos para relocalización en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles . . . . .	105
7.36. Pérdida de la estimación en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles . . . . .	105
7.37. Primer fotograma con extracción de características en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles . . . . .	105

7.38. Búsqueda de nuevos puntos para relocalización en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles . . . . .	105
7.39. Seguimiento de características final en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles . . . . .	105
7.40. Inicio de estimación en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles . . . . .	106
7.41. Seguimiento de la estimación en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles . . . . .	106
7.42. Finalización de la estimación en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles . . . . .	106
7.43. Seguimiento de características robustas para triangulación en prueba de cajas 1 . . . . .	107
7.44. Primer fotograma con extracción de características en prueba de cajas 1 . . . . .	107
7.45. Salida terminal con el mapa de triangulación en prueba de cajas 1 . . . . .	107
7.46. Seguimiento de características durante el vídeo de prueba de cajas 1 . . . . .	108
7.47. Seguimiento de características al desplazar la cámara a la derecha durante el vídeo de prueba de cajas 1 . . . . .	108
7.48. Inicio de la estimación en prueba de cajas 1 . . . . .	108
7.49. Final de la estimación en prueba de cajas 1 . . . . .	108
7.50. Seguimiento de características robustas para triangulación en prueba de cajas 2 . . . . .	109
7.51. Primer fotograma con extracción de características en prueba de cajas 2 . . . . .	109
7.52. Salida terminal con el mapa de triangulación en prueba de cajas 2 . . . . .	109
7.53. Extracción de características a la mitad de la prueba de cajas 2 . . . . .	110
7.54. Extracción de características al final de la prueba de cajas 2 . . . . .	110
7.55. Inicio de la estimación en prueba de cajas 2 . . . . .	110
7.56. Estimación en la mitad de la prueba de cajas 2 . . . . .	110
7.57. Final de la estimación en prueba de cajas 2 . . . . .	110
7.58. Gráfica tamaño requerido para almacenar los eventos según umbrales distintos en MB . . . . .	113
7.59. Gráfica tamaño requerido para almacenar los eventos según distintas resoluciones en MB . . . . .	114
7.60. Gráfica con la proporción de píxeles de media por evento en cada vídeo . . . . .	115
7.61. Gráfica con los tamaños equivalentes a un evento en cada vídeo según el formato empleado . . . . .	116
7.62. Gráfica con los tamaños equivalentes a un evento en cada vídeo según el formato empleado . . . . .	117

7.63. Tiempo empleado en generar eventos de un vídeo haciendo uso de la CPU . . . . .	118
7.64. Tiempo empleado en generar eventos de un vídeo con uso de GPU . . . . .	119
7.65. Tiempo total empleado por V2E haciendo uso de la CPU (azul) o de la GPU (verde) . . . . .	120
7.66. Tiempos de la fase de interpolación en distintas resoluciones . . . . .	120
7.67. Tiempos de la fase de simulación de eventos en distintas soluciones . . . . .	121
7.68. Tiempos totales del simulador V2E con distintas resoluciones . . . . .	121
7.69. Calibración en vídeos a 346x260 píxeles grabados con el iPhone SE . . . . .	122
7.70. Calibración en vídeos a 720p grabados con el iPhone SE . . . . .	122
7.71. Ejemplo de malos parámetros de calibración con respecto a la resolución del vídeo . . . . .	123
7.72. Ejemplo de malos parámetros de calibración con respecto al punto central de la cámara . . . . .	123
7.73. Fotograma de la prueba con muñeco en escritorio, en el intervalo indicado por los parámetros “start_time_s” y “stop_time_s” indicados en la Tabla 7.3.1 . . . . .	124
7.74. Eventos recogidos para la reconstrucción 3D de la prueba con muñeco en escritorio . . . . .	124
7.75. Mapa de confianza del reconstructor EMVS en la estimación de la profundidad de cada evento de la prueba con muñeco en escritorio . . . . .	124
7.76. Mapa de profundidad en escala de grises de la prueba con muñeco en escritorio . . . . .	125
7.77. Mapa de profundidad a color de la prueba con muñeco en escritorio . . . . .	125
7.78. Mapa de profundidad a color de la prueba con muñeco en escritorio con 3 capas de profundidad . . . . .	126
7.79. Nube de puntos generada con 3 capas de profundidad de la prueba con muñeco en escritorio . . . . .	126
7.80. Eventos recogidos para la reconstrucción 3D de la prueba con muñeco en escritorio a una resolución de 1280x720 píxeles . . . . .	127
7.81. Mapa de profundidad a color de la prueba con muñeco en escritorio a una resolución de 1280x720 píxeles . . . . .	128
7.82. Fotograma de la prueba con objetos pequeños en escritorio, en el intervalo indicado por los parámetros “start_time_s” y “stop_time_s” de la Tabla 7.3.1 . . . . .	128
7.83. Eventos recogidos para la reconstrucción 3D de la prueba con objetos pequeños en escritorio . . . . .	129
7.84. Mapa de profundidad a color de la prueba con objetos pequeños en escritorio . . . . .	129

7.85. Eventos recogidos con parámetros extra para la reconstrucción 3D de la prueba con objetos pequeños en escritorio . . . . .	130
7.86. Mapa de profundidad a color extraído con parámetros extra de la prueba con objetos pequeños en escritorio . . . . .	130
7.87. Mapa de confianza en la estimación de la profundidad de cada evento en la prueba con objetos pequeños en escritorio . . . . .	130
7.88. Mapa de profundidad a color con 200 capas de la prueba con objetos pequeños en escritorio . . . . .	130
7.89. Salida terminal creación nube de puntos antes y después de la reducción de ruido . . . . .	131
7.90. Fotograma correspondiente a la utilizada para la reconstrucción 3D de la prueba cajas 1 . . . . .	132
7.91. Eventos recogidos para la reconstrucción 3D de la prueba de cajas 1 . . . . .	133
7.92. Mapa de profundidad a color de la prueba de cajas 1 . . . . .	133
7.93. Nube de puntos de la prueba de cajas 1 . . . . .	133
7.94. Fotograma en el intervalo seleccionado para la reconstrucción 3D de la prueba con cajas 2 . . . . .	134
7.95. Eventos recogidos para la reconstrucción 3D de la prueba de cajas 2 . . . . .	134
7.96. Mapa de profundidad a color de la prueba de cajas 2 . . . . .	134
7.97. Eventos recogidos con menor ruido para la reconstrucción 3D de la prueba de cajas 2 . . . . .	135
7.98. Mapa de profundidad a color con menor ruido de la prueba de cajas 2 . . . . .	135
7.99. Entorno grabado para la prueba “Shapes” de la universidad de Zurich . . . . .	136
7.100 Eventos recogidos para la reconstrucción 3D para la prueba “Shapes” de la universidad de Zurich . . . . .	136
7.101 Mapa de profundidad a color para la prueba “Shapes” de la universidad de Zurich . . . . .	136
7.102 Nube de puntos para la prueba “Shapes” de la universidad de Zurich . . . . .	136
7.103 Nube de puntos para la prueba “Shapes” de la universidad de Zurich . . . . .	136
7.104 Entorno grabado para la prueba “Boxes” de la universidad de Zurich . . . . .	137
7.105 Mapa de profundidad a color para la prueba “Boxes” de la universidad de Zurich . . . . .	137
7.106 Entorno grabado para la prueba “Slider_depth” de la universidad de Zurich . . . . .	138
7.107 Eventos recogidos para la reconstrucción 3D para la prueba “Slider_depth” de la universidad de Zurich . . . . .	138

7.108	Mapa de profundidad a color para la prueba “Slider_depth” de la universidad de Zurich . . . . .	139
7.109	Nube de puntos para la prueba “Slider_depth” de la universidad de Zurich con una vista lateral izquierda . . . . .	139
7.110	Entorno grabado para la prueba “Dynamic” de la universidad de Zurich . . . . .	139
7.111	Eventos recogidos para la reconstrucción 3D para la prueba “Dynamic” de la universidad de Zurich . . . . .	139
7.112	Mapa de profundidad a color para la prueba “Dynamic” de la universidad de Zurich . . . . .	140
7.113	Nube de puntos para la prueba “Dynamic” de la universidad de Zurich . . . . .	140
7.114	Nube de puntos para la prueba “Dynamic” de la universidad de Zurich con una toma lateral derecha . . . . .	140

# Índice de cuadros

2.1. Coste en recursos Hardware . . . . .	50
2.2. Coste en recursos Software . . . . .	50
2.3. Coste total para el proyecto . . . . .	51
4.1. Caso de uso 01 . . . . .	62
4.2. Caso de uso 02 . . . . .	62
4.3. Caso de uso 03 . . . . .	62
4.4. Caso de uso 04 . . . . .	63
4.5. Caso de uso 05 . . . . .	63
4.6. Caso de uso 06 . . . . .	63
7.1. Parámetros SVO . . . . .	95
7.2. Lista de parámetros SVO modificados . . . . .	97
7.3. Tamaño de archivo con distinto umbral en la generación de eventos . . . . .	112
7.4. Tamaño de archivo con distinta resolución en la generación de eventos . . . . .	113
7.5. Tamaño requerido para almacenar la información equivalente a un evento en cada prueba . . . . .	116
7.6. Parámetros de configuración del EMVS para la prueba con muñeco en escritorio . . . . .	123
7.7. Parámetros de configuración del EMVS para la prueba con objetos pequeños en escritorio . . . . .	129
7.8. Parámetros de configuración añadidos adicionalmente para la prueba con objetos pequeños en escritorio . . . . .	129
7.9. Parámetros de reducción de ruido para la nube de eventos para la prueba con objetos pequeños en escritorio . . . . .	131
7.10. Parámetros de configuración del EMVS para la prueba cajas 1 . . . . .	132
7.11. Parámetros de configuración del EMVS para la prueba cajas 2 . . . . .	134
7.12. Parámetros de configuración del EMVS para la prueba “Shares” de la universidad de Zurich . . . . .	136
7.13. Parámetros de configuración del EMVS para la prueba “Boxes” de la universidad de Zurich . . . . .	137

7.14. Parámetros de configuración del EMVS para la prueba “Sli-	
der_depth” de la universidad de Zurich . . . . .	138
7.15. Parámetros de configuración del EMVS para la prueba “Dy-	
namic” de la universidad de Zurich . . . . .	140

# Capítulo 1

## Introducción

La reconstrucción 3D constituye aquellas técnicas capaces de representar diferentes entornos en tres dimensiones a partir de una información, ya sea directamente desde unos datos de profundidad ya almacenados u obtenidos a través de láseres o con ultrasonidos, o bien sin interferir con el entorno, interpretando la información proporcionada por imágenes para realizar la reconstrucción 3D.

Dicha reconstrucción 3D tiene una gran variedad de campos de aplicación como la medicina (implantes o reconstrucciones en huesos hechos a medida)[\[1\]](#), realidad aumentada [\[2\]](#), entretenimiento en los sectores de los videojuegos y el cine (reconstrucciones de ciudades o lugares históricos para su uso en el mundo virtual)[\[3\]](#), planificación urbana [\[4\]](#), o reconocimiento de objetos en 3D [\[5\]](#). En este último caso, con la irrupción de las impresoras 3D en los últimos años, la capacidad de reconstruir en 3D un objeto real para replicarlo ha conseguido que cobre aún más importancia si cabe.

Pero el campo en el que su importancia es esencial es la robótica y la visión por computador, siendo uno de los problemas más estudiados el del SLAM (Simultaneous Localization And Mapping, o localización y mapeado simultáneos). De todos modos, a pesar de ser un campo muy desarrollado, siguen dándose problemas a resolver en lo que respecta al localizado y mapeado en entornos dinámicos, tales como en terrenos donde transcurre un desastre natural o algún tipo de accidente en el que el mapeado del entorno debe ser reconstruido al instante por los constantes cambios sufridos en la escena o la presencia de objetos que se mueven.

Para este tipo de problemas deberíamos tener un procesamiento que pueda llevarse a cabo en tiempo real, lo cual es muy complejo debido a que la mayoría de métodos tradicionales de reconstrucción 3D requieren de unos

pasos previos muy precisos y con una gran complejidad computacional antes de captar los datos para la reconstrucción 3D.

Además la información se procesa una vez obtenida del dron/robot que usemos, sin la capacidad de recibir los resultados del entorno 3D en tiempo real, y también necesitando de un largo y complejo procesamiento por los algoritmos muy costosos en el cálculo del modelo 3D.

Los primeros intentos de su implementación hacían uso del filtro de Kalman [6] con distintas variaciones del mismo para mejorar los resultados obtenidos, aunque incrementando la complejidad computacional [7], [8]. Este problema puede solucionarse a partir de la combinación de distintos sensores como giróscopos y acelerómetros o con un sensor global de posicionamiento (GPS), pero también se puede resolver a partir de cámaras, con la variante de SLAM Visual [9].

Gracias a los grandes avances en la tecnología y con un hardware mucho más capaz en cuanto a rendimiento, sumado al desarrollo de técnicas de paralelización y las eficientes soluciones desarrolladas, han hecho mejorar el SLAM basado en fotogramas clave con visión monocular. El LSD-SLAM (Large-Scale Direct SLAM) [10], se ha convertido en un pilar fundamental en el estado del arte mejorando la estimación de la posición de la cámara en lugares abiertos. El método usa toda la información de las imágenes, consiguiendo una mayor precisión y solidez en entornos con poca textura como en interiores y una reconstrucción 3D más densa. Como contrapartida, al usar toda esa información es computacionalmente exigente y tarda demasiado para llegar a utilizarse sin el uso de una GPU potente y menos aún para alcanzar el procesamiento en tiempo real.

## 1.1. Motivación

La motivación de este trabajo es buscar alternativas a los métodos tradicionales de reconstrucción 3D, donde se use la información captada por los sensores de eventos y obtener unos resultados similares si cabe, aprovechando la reducción en la cantidad de datos a procesar. Esto ayudaría a reducir los tiempos de procesamiento para generar la simulación del entorno 3D y minimizar los recursos computacionales necesarios, para conseguir un rendimiento en tiempo real, lo que a su vez nos permitiría su uso en por ejemplo, sistemas empotrados para drones.

En este trabajo exploramos el uso de unos sensores biológicamente inspirados llamados retinas artificiales. Estos sensores capturan solamente la información del cambio de intensidad en cada píxel del sensor en lugar del valor

absoluto de la intensidad. Nuestro objetivo final sería el de estimar la profundidad en la imagen para la reconstrucción 3D usando este tipo de sensores. A su vez, su uso nos permitiría aprovechar la menor carga computacional al trabajar con una menor cantidad de datos. De esta forma, podríamos llegar a realizar el procesamiento en tiempo real y así poder utilizarlo en aplicaciones como las mencionadas: para el movimiento autónomo de un dron o la identificación de objetos en un entorno 3D.

Además, para obtener la información de entrada se hará uso de un simulador de eventos, con la idea de una mejor comprensión y análisis de las ventajas dadas en la información aportada por un sensor de eventos frente a los vídeos con píxeles en formato RGB. Aprovechando así la posibilidad de emplear técnicas de visión por computador clásicas al disponer de fotogramas grabados con una cámara tradicional, como por ejemplo para la estimación de la posición de la cámara, necesario para llevar a cabo la reconstrucción 3D.

Para el trabajo se pretende hacer uso de varios sistemas distintos, todos interactuando entre ellos con el objetivo final de obtener la reconstrucción 3D y con la posibilidad de la ampliación de más componentes con distintas funcionalidades dentro del mismo sistema. Por ello se hará uso del middleware ROS (Robot Operating System) [11], ideal para desarrollar el proyecto y así poder usar las distintas facilidades que otorga para el tratamiento de datos, más la modularización de los distintos sistemas para tener un mismo flujo de trabajo en la misma aplicación.

## 1.2. Estado del arte

### 1.2.1. Métodos usados para la conducción autónoma de drones o SLAM en entornos dinámicos

En la actualidad está muy extendido el uso de UAVs (Vehículos aéreos no tripulados) o drones para realizar distintas tareas, como vigilancia, transporte o para emergencias, ya sea para búsqueda y rescate, detección de incendios, localización de cuerpos, etc... Las técnicas usadas para su correcto funcionamiento en cada una de estas situaciones vienen determinadas principalmente por los sensores a bordo del propio dron, como receptor GPS, una cámara de alta resolución con capacidad de transmisión en tiempo real, sistemas anti-collisiones como sensores de cercanía, o radares 3D. Dependiendo del uso que queramos hacer del dron o los elementos que tenga disponibles hay que realizar distintos acercamientos mediante software para solucionar estas tareas. Por ejemplo, los sensores de proximidad por infrarrojos no son adecuados para su uso en exteriores debido a la radiación infrarroja ambiental. Del mismo modo, en entornos interiores, el GPS no está disponible para

la localización y las cámaras no pueden utilizarse eficazmente en condiciones de iluminación baja/variable.

En la literatura científica, encontramos algunos métodos que se aproximan a resolver este problema a través de la inteligencia artificial. Por ejemplo, el trabajo [12] que busca una aproximación de la conducción autónoma de drones a partir de aprendizaje por refuerzo (Reinforcement Learning) en entornos conocidos para aplicarlo en desarrollos futuros a entornos desconocidos o dinámicos (por ejemplo, en desastres naturales). Otro de los trabajos que encontramos trata el problema del movimiento autónomo de drones en interiores con obstáculos y sin información GPS [13]. Este trabajo usa un algoritmo simple en el que se evitan los obstáculos, que son desconocidos, a la vez que se aproximan al destino deseado con el uso de láseres infrarrojos y un sonar para estimar la altitud del propio dron.

Una de las posibles aplicaciones de estos desarrollos puede ser la de la ayuda para trabajadores de servicios de emergencia como bomberos, para entrar en terrenos peligrosos o edificios en situaciones límite, en los que no se conoce bien el estado del entorno. Para ellos, un sistema de localización y mapeado de este tipo sería vital para el apoyo en operaciones de búsqueda y rescate. Esto buscan en el trabajo [14], en el que se fusionan dos métodos desarrollados para SLAM con distintas herramientas. En primer lugar con un LiDAR (Detección y alcance de láser por imagen), que es un dispositivo que lanza un láser a un objeto y mide la distancia que hay con el tiempo que transcurre entre la emisión del propio láser y la recepción de la señal reflejada. Y en segundo lugar con una cámara para el visual SLAM que consiste en dos pasos: 1º Buscar la correspondencia de los puntos característicos captados de la imagen actual con los puntos del mapa del entorno generado en el último fotograma y 2º actualizar y optimizar el mapa del entorno y la localización actual.

Se fusionan ambos métodos, y se les suma el uso de una IMU y GPS, como apoyo a ambas técnicas SLAM para validar los resultados obtenidos, confeccionando este sistema equipable SLAM que se aprecia en la Figura [1.1].

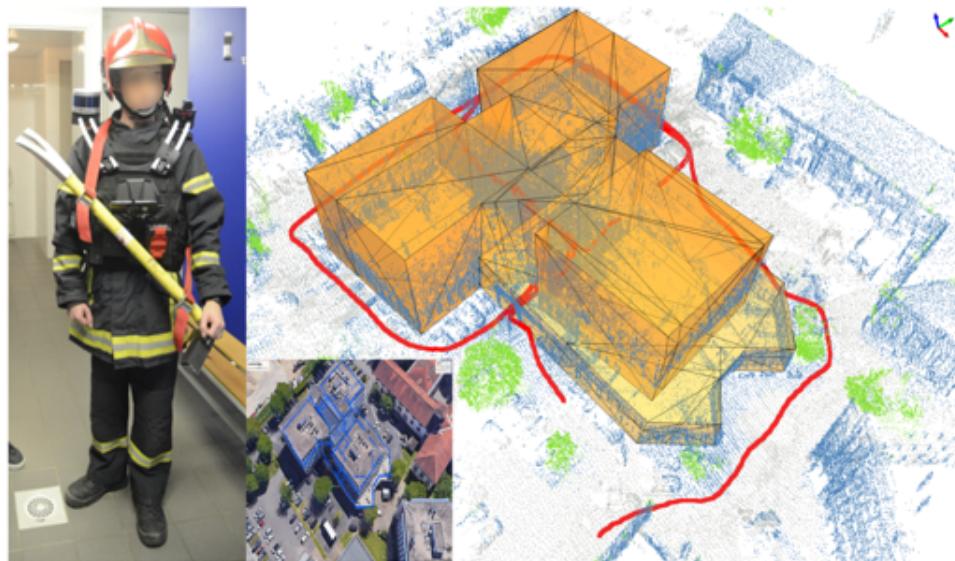


Figura 1.1: SLAM basado en un agente. Izquierda: Agente con el sistema equipado. Centro: Imagen de Google Earth del edificio Derecha: Mapeado 3D sin conexión del entorno del edificio tras moverse el agente por el recorrido en rojo.

### 1.2.2. Cámara de eventos

Como mencionamos anteriormente, en este proyecto usaremos las retinas artificiales como sensor visual, para aprovechar las ventajas que proporcionan frente a las cámaras convencionales. Por un lado, tratando de llevar a cabo el procesamiento en tiempo real gracias a la compresión automática de la escena que hacen de manera natural al registrar únicamente cambios en la intensidad. Por otro, conseguir buenos resultados en la estimación de la profundidad y la posición de la cámara en la escena.

Estos sensores con base biológica, también referidos como cámaras de eventos [15], [16], [17] , presentan un nuevo paradigma sobre el modo en el que se adquiere y procesa la información visual. Cada píxel de la cámara funciona independiente al resto, controlando continuamente el nivel de intensidad del brillo en el mismo e informando de cualquier cambio en el mismo con una precisión de microsegundos, generando en ese píxel un “evento”. Estos eventos vienen dados con la siguiente estructura:  $e = (x,y,t,p)$  y se generan si en el pixel con posición  $(x,y)$  se genera un cambio de intensidad que supere un umbral (normalmente de un 10 %-15 % de cambio relativo), con una marca de tiempo  $t$  en la que se ha producido y una polaridad  $p$ , que será  $\pm 1$  marcando el signo del cambio en la intensidad.

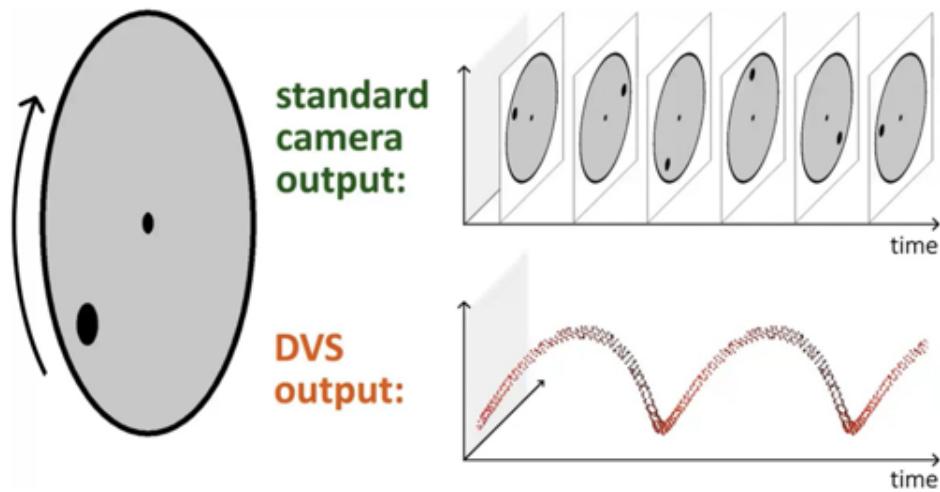


Figura 1.2: Comparativa entre cámaras tradicionales y sensores DVS, captura de la animación que podemos encontrar en <https://youtube.com/watch?v=LauQ6LWTkxM>

Para comprender mejor la diferencia entre las cámaras tradicionales y estos sensores de eventos se muestra la Figura 1.2. En ella, gira un disco con un círculo negro y se muestra la salida de los dos dispositivos. En la cámara estándar observamos cómo la información que capta es la imagen del propio disco a una frecuencia fija, enviando información insuficiente para seguir de forma precisa el movimiento o en otros casos es redundante al no existir movimiento. Además, debido a la velocidad a la que se registra la imagen, se produce emborronamiento por el movimiento de giro. Por el contrario, en las retinas artificiales no están fijados a una frecuencia de fotogramas, si no que cada vez que existen datos (cambio de intensidad en un pixel) se genera una salida con una latencia de solo unos microsegundos entre eventos consecutivos para el mismo pixel. Por lo tanto, no se producirá ninguna salida en ausencia de movimiento en la escena. Además, debido a la alta velocidad a la que se disparan los eventos (del orden de 1000-10000 veces más rápidos que las cámaras convencionales), tampoco se producirá el emborronamiento.

### Tipos:

Estos sensores bioinspirados o retinas artificiales surgieron por primera vez en la tesis [18] en los años 1986-1992, reconocida con el prestigioso premio Clauser del instituto tecnológico de California (Caltech). En la siguiente década se fueron desarrollando mejores modelos de estas retinas artificiales, comenzando por la cámara de eventos DVS [16], con un

diseño en el que el fotorreceptor de tiempo continuo está acoplado a un circuito de lectura que se reinicia cada vez que se muestrea el píxel. Esto mejora el rendimiento y da la posibilidad de resolver muchas aplicaciones con el uso de los eventos generados por DVS, pero a pesar de ello, se hizo evidente la necesidad de combinar esta información de eventos con otro tipo de sensores, como detectores de brillo absoluto en la imagen, para poder realizar alguna de esas aplicaciones.

El sensor de imagen basado en tiempo asíncrono (ATIS) [19] es una de los avances para solucionar el problema, contiene cada píxel un subpíxel DVS junto a otro subpíxel que lee la intensidad luminosa para dar el valor en escala de grises, solo en los píxeles que se produce un cambio de intensidad detectado por el subpíxel DVS y así actualizan el valor de intensidad de la imagen.

Pero estos sensores tienen desventajas, como que el tamaño del píxel es al menos el doble que los de un píxel DVS, provocando un mayor coste en la producción de estos sensores, además que en entornos oscuros el tiempo entre dos eventos puede ser muy largo, llegando a solapar los valores de intensidad absoluta por la llegada de nuevos eventos en mitad del proceso.

Otro de los acercamientos se hizo con los sensores DAVIS [20], que combina en un mismo píxel la tecnología DVS y la del sensor de imagen convencional APS. La ventaja sobre ATIS es el tamaño del píxel, siendo este menor ya que el fotodiodo es compartido y solamente añade un 5 % de área más al píxel DVS. Los sensores APS pueden ejecutarse de forma síncrona, y como pasaba en las cámaras tradicionales, da información redundante en caso de no cambiar la escena. Como estas soluciones DAVIS y ATIS hacen uso de los píxeles DVS, al conjunto de todos estos sensores se les denomina comúnmente “sensores DVS” para referirse a ellos sin importar cuál sea.

### Disponibilidad:

Estas cámaras de eventos tienen unas características diferentes a otros sensores de imágenes con tecnología CMOS, por ello se requiere de una estandarización de estos sensores DVS para poder realizar investigaciones en el campo. Desde la creación de la primera cámara de eventos se intentó ir desarrollando mejores versiones en las que se aumentara la velocidad de captura, se incrementara la resolución espacial o se añadieran nuevas funcionalidades, como por ejemplo una salida de imagen en escala de grises que se incorporó en sensores DAVIS y ATIS o la integración de un IMU (Unidad de Medición Inercial)<sup>1</sup>.

---

<sup>1</sup>[https://es.wikipedia.org/wiki/Unidad\\_de\\_medici%C3%B3n\\_inercial](https://es.wikipedia.org/wiki/Unidad_de_medici%C3%B3n_inercial)

Su poca resolución espacial es otra de las limitaciones en cuanto a la aplicación de estos sensores. Sin embargo, la mayoría de los trabajos publicados están desarrollados con el sensor DVS128 que cuenta con una resolución de únicamente 128x128 píxeles y consiguen resultados muy destacables.

El coste de la cámara está en el rango de entre los 4000 euros y los 10000 euros dependiendo de si viene únicamente con el sensor DVS o incluya también sensor de imagen como en la DAVIS. Actualmente esto supone un problema para la adquisición de esta tecnología, como ya pasaba por ejemplo en los comienzos de las cámaras térmicas. Se estima que este precio bajará significativamente una vez entre en producción masiva, como ya se puede apreciar en el producto de “Samsung SmartThings Vision” [21], que contiene un sensor de eventos usado para videovigilancia en el hogar y tiene un valor de 100 euros.

### Ventajas:

- Resolución temporal alta: El sensor es capaz de detectar cambios en la intensidad y asociarlos una marca de tiempo con precisión del orden de microsegundos. Lo cual hace que reduzcan en consideración el emborronamiento por movimiento que sufren las cámaras tradicionales. Además, esta cualidad hace a estos sensores muy adecuados para aplicaciones en las que los objetos se mueven a altas velocidades.
- Baja latencia: Cada píxel es procesado de forma independiente y transmite la información conforme detecta el cambio, sin necesidad de esperar a un proceso reloj común, como ocurre en los sensores convencionales síncronos.
- Baja potencia: La potencia que se requiere para hacer funcionar estos sensores solo se necesita cuando se produce cambio de intensidad. En sistemas empotrados con sensores de eventos se ha medido que el coste es de 100 mW o menos.
- Alto rango dinámico (HDR): Al superar con creces el alto rango dinámico de las cámaras tradicionales (120dB frente a 60 dB en cámaras de alta calidad) hace posible que los sensores DVS puedan mostrar la información durante todo el día, adaptándose a situaciones muy oscuras como también muy luminosas.

**Desafíos:**

Se necesitan nuevas técnicas para aprovechar las ventajas de estos sensores debido a la naturaleza atípica de la representación de datos de los mismos para las técnicas de visión por computador actuales, que llevan a los siguientes desafíos.

Uno de los desafíos que nos encontramos son los tiempos en los que se generan las salidas. Al contrario que en las cámaras tradicionales donde el tiempo en el que viene cada fotograma es fijo, en los sensores de eventos es asíncrono. Además, la información obtenida por estos sensores es dispersa a diferencia de las tradicionales, donde obtenemos información en todos los píxeles de la imagen (100% densa).

La información que transmiten los sensores DVS sigue una filosofía totalmente diferente, tal que si en un anterior instante de tiempo existía un valor de intensidad en el brillo de la imagen menor o mayor al actual, registran ese cambio, informando por tanto únicamente cuando sucedan cambios de intensidad. En cámaras tradicionales, se capturan valores de intensidad en escalas de grises, lo cual cambia la forma de tratar esta información. Pero además, esta forma de percibir el entorno comprime la información en la escena, reduciendo la transmisión de información redundante. Además, esta compresión es inteligente porque los cambios en la intensidad se producen en los bordes de los objetos o las texturas.

### 1.2.3. Simuladores de eventos

Estos sensores de eventos proporcionan distintas ventajas que se pueden explotar para distintas técnicas de visión por computador, como la reconstrucción 3D de este trabajo, pero tienen el inconveniente de ser caros, además de ser recientes en el estado del arte y no estar tan desarrollado en distintos campos.

Por ello, las salidas de estas cámaras de eventos pueden simularse desde imágenes usando una técnica de Transfer Learning (aprendizaje por transferencia), que consiste en trasladar el conocimiento de cómo obtienen los resultados estas cámaras de eventos, e intentar replicarlo siguiendo estos pasos en vídeos convencionales para obtener eventos simulados. Así podemos aprovechar y utilizar los fotogramas tradicionales del vídeo original, siendo más sencillo la integración de otras técnicas desarrolladas para imágenes convencionales en el proyecto.

Las herramientas Event Camera Dataset and Simulator [17] y el más reciente ESIM [22] son dos de los sistemas que pueden usarse para la generación

de estos eventos a partir de vídeos.

Una extensión de ESIM llamada rpg\_vid2e maneja ESIM a partir de fotogramas de un vídeo interpolados [23]. La herramienta rpg\_vid2e simula eventos DVS ideales con buena iluminación, pero no eventos DVS realistas con mala iluminación, que es un caso de uso importante para los sensores DVS.

El sistema v2e es un paso hacia la incorporación de un modelo DVS más realista en la simulación de eventos. Al permitir el control explícito del ruido, v2e permite la generación de eventos en vídeos que cubren una gran gama de condiciones de iluminación. Además su sencillez de uso y los distintos archivos útiles que aportan como salida nos hacen decantarnos por el simulador de eventos v2e.

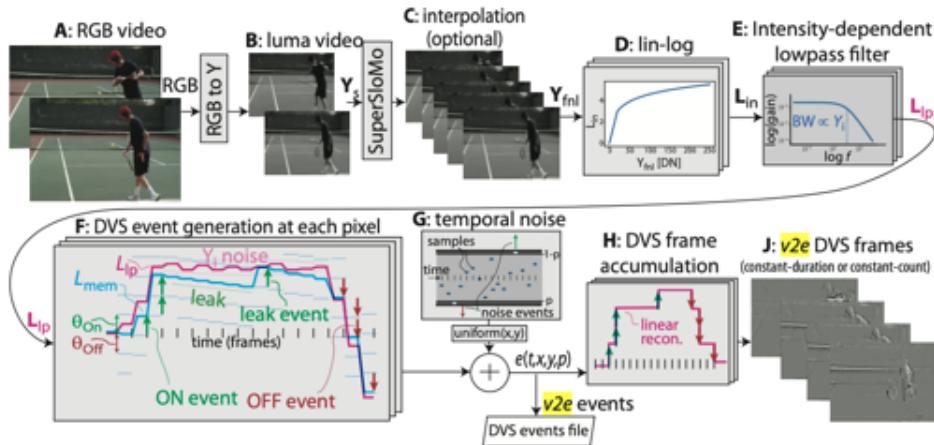


Figura 1.3: Pasos del v2e para simular eventos como los sensores DVS

El proceso que sigue este simulador v2e se puede apreciar en la Figura 1.3 desde la entrada del vídeo en escala RGB hasta la generación de los fotogramas DVS simulados por v2e.

**A-B: Conversión del color a luma:** v2e comienza con la conversión del vídeo de entrada al formato deseado para poder trabajarla, convirtiendo los fotogramas de color a fotogramas luma, un formato que sigue la recomendación BT. 709 de la ITU-R se usa la fórmula para cada pixel tal que  $Y' = 0.2126 R' + 0.7152 G' + 0.0722 B'$  [24], dejando una imagen parecida al equivalente del blanco y negro de la imagen original. Una vez hecho esto se redimensiona en caso de que se especifique un tamaño distinto para el ancho

y alto con respecto al vídeo original.

**C: Simulación cámara lenta:** Estos fotogramas se interpolan de forma opcional con una red neuronal llamada Super-SloMo [25], para aumentar la resolución temporal del vídeo de entrada. Super-SloMo predice el flujo óptico bidireccional a partir de dos fotogramas luma consecutivos, utilizando para interpolar linealmente nuevos fotogramas entre los dos fotogramas originales. Para una mejor estimación del flujo entre los fotogramas luma, se volvió a entrenar a Super-SloMo con la base de datos Adobe240FPS tras convertir sus fotogramas RGB en fotogramas luma.

**D: Cambio de función lineal a logarítmica:** El método usado para generar los eventos consiste en detectar los cambios en la intensidad de la imagen, que viene representada normalmente de forma lineal en los vídeos digitales. No como en los sensores DVS, que tienen una representación logarítmica para detectar los cambios de intensidad y captar eventos. Por defecto, la visión por computador usa una representación de 8-bits, que equivale a un rango dinámico limitado de 255. Para hacer frente a esta limitación se hace un mapeo lineal-logarítmico entre los valores lineales de un fotograma y el valor de la intensidad logarítmica como en la Figura 1.3, reduciendo el ruido en la salida simulada del DVS.

**E: Ancho de banda del fotorreceptor dependiente de la intensidad:** Como los píxeles de un sensor DVS real tienen un ancho de banda finito, se le pasa un filtro de paso bajo que filtra los valores de entrada logarítmicos. Este filtro modela la respuesta del sensor DVS en condiciones de baja iluminación, tal y como comentaba en las ventajas de este simulador anteriormente.

Se definen los umbrales en los píxeles para generar eventos positivos y negativos, con un valor normalmente similar y entre 0.1 y 0.4, siendo con signo positivo para el caso de los eventos positivos y con signo negativo para eventos negativos. Esto se traduce a que el rango típico de umbrales en los sensores DVS para el cambio de intensidad es de entre un 10 % a un 50 %. Estos umbrales de eventos son adimensionales y representan un cambio de intensidad relativa, es decir, un umbral sobre el cambio de la intensidad con respecto al valor memorizado anteriormente. Estos cambios de intensidad relativa se producen por los cambios de reflectancia de la escena, por lo que esta representación es útil para producir eventos que sean representativos del vídeo de entrada.

**F: Modelo de generación de eventos:** Suponemos que el píxel tiene un valor de brillo memorizado en intensidad logarítmica ( $B_{mem}$ ) y que el nuevo valor de brillo con el filtrado de paso bajo es  $B_{pb}$ . El modelo genera entonces una cantidad entera con signo  $N_e$  de eventos positivos ON o nega-

tivos OFF a partir del cambio  $\Delta B = B_{pb} - B_{mem}$  donde  $N_e = \Delta B / \text{umbral}$ . Si  $\Delta L$  es un múltiplo de los umbrales ON y OFF, se generan múltiples eventos DVS. El valor de luminosidad memorizado se actualiza por  $N_e$  múltiplos del umbral.

**G: Ruido temporal:** La naturaleza cuántica de los fotones da como resultado que se disparen eventos por ruido, a bajas intensidades de luz el efecto de este ruido aumenta dramáticamente, dando como resultado eventos por ruido equilibrados en positivos y negativos a una frecuencia superior a 1Hz por píxel. El sistema v2e modela el ruido temporal utilizando un proceso que genera eventos de ruido temporal positivos y negativos, ajustándose a una frecuencia de eventos de ruido  $R_n$  (por defecto 1 Hz).

**Tiempos de los eventos:** Las marcas de tiempo de los fotogramas interpolados son discretas. Dados dos fotogramas interpolados consecutivos, las marcas de tiempo de los eventos se distribuyen uniformemente entre el tiempo del fotograma  $n$  y el tiempo del fotograma  $n+1$ .

#### 1.2.4. Estimación de la posición de la cámara

La posición y orientación de la cámara es otro de los datos necesarios para llevar a cabo la tarea de la reconstrucción 3D, el estudio de esta información se denomina odometría, y en caso de obtenerse a través de imágenes, odometría visual, que consiste en dar una posición en el espacio, normalmente definida a partir de unas coordenadas  $(x,y,z)$  que suponen un punto en el entorno, además de la orientación a la que apunta la cámara, representado con un cuaternio  $(x,y,z,w)$ . Los datos son relativos a una posición y orientación de inicio definida al comienzo de la estimación de la posición de la cámara, y a partir de ella se estiman las distintas posiciones, como por ejemplo se observa en la Figura 1.4, donde se pasa de la posición y orientación iniciales  $F_1$ , a la posición y orientación posterior  $F_2$ .

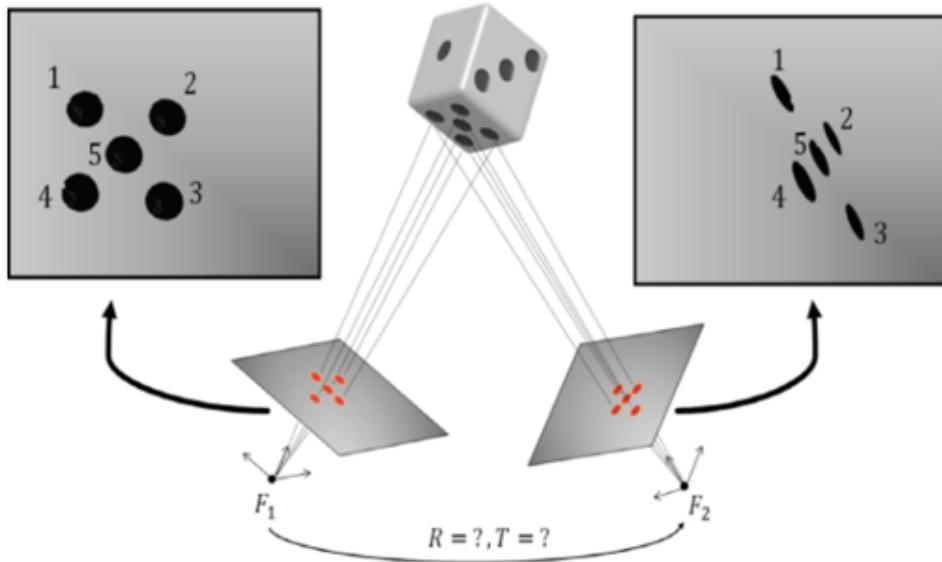


Figura 1.4: Ejemplo de la posición y movimiento de la cámara extraída

Esta es una parte esencial para la reconstrucción 3D, necesaria para el cálculo de la profundidad de las distintas zonas de la imagen, trazando rayos desde un mismo punto de la escena a cada una de las posiciones de la cámara estimadas, para así localizar el lugar en tres dimensiones donde se encuentra ese punto de la escena, como se explicará más a fondo en la siguiente sección [1.2.5](#).

Lo ideal sería tener una IMU para captar el movimiento y posición del sensor DVS, el cual contiene una combinación de acelerómetros y giroscopios que son capaces de medir la velocidad, orientación y fuerzas gravitacionales de un aparato. Normalmente son usadas para maniobrar desde vehículos aéreos no tripulados hasta incluso naves espaciales y es el componente principal en los sistemas de navegación de misiles autoguiados. En caso de no disponer de una IMU, hay que hacer uso de otros sistemas que sean capaces de determinar la posición del sensor DVS con la información disponible, que en este caso son los eventos.

El estado del arte en cuanto a la estimación de la posición del sensor a través de la información de los eventos es muy escaso, en estos sistemas se usan unos acercamientos totalmente distintos al resto de técnicas desarrolladas para el problema del SLAM, debido a la diferente información disponible captada por un sensor de eventos y una cámara tradicional, por lo que se debió investigar de cero para encontrar el modo de aprovecharse de las cámaras de eventos.

Naturalmente ambos problemas de estimación de posición y movimiento de la cámara junto a la reconstrucción 3D vienen abordados en SLAM, por lo que el estado del arte comenzó a centrarse en resolver este problema para los sensores de eventos y comenzaron definiendo los tres pilares fundamentales a abordar para el problema del SLAM, la dimensionalidad del problema, el tipo de movimiento realizado y el tipo de escena que está grabando.

Empezaron por resolver los problemas más sencillos y paso a paso ir aumentando la complejidad del problema, en cuanto al tipo de movimiento realizado se estimaron primero movimientos restringidos como rotaciones o en un mismo plano, ambos son 3-DOF o 3 grados de libertad, movimientos que incluyen las rotaciones y movimientos de cámara en un mismo punto sin cambiar de posición en el espacio para mirar hacia los lados, arriba y abajo, pasando después a los más complejos 6-DOF que ya tienen en cuenta la posición de la cámara. Para aproximarse al problema del tipo de escena, comenzaron con escenas artificiales en 2D con un alto contraste antes de pasar a escenas naturales en 3D. Otra de las ayudas que se tomaron es abordar el problema con ayuda de otras técnicas más conocidas como con imágenes a escala de grises o con un sensor RGB-D [26], pero esto conlleva arrastrar las desventajas que se producen al usar fotogramas estándar, como por ejemplo la latencia o el desenfoque de movimiento.

Un primer trabajo [27] consiguió mediante un algoritmo que pasaba por mensajes la información de los eventos a una red que estimaría el movimiento aunque estaba restringido a rotaciones. Le siguió el trabajo [28] que podía estimar los movimientos de la cámara en paralelo a la escena 2D que está grabando y con la limitación añadida de ser blanco y negro con líneas en vez de una escena natural.

Se pasó a una estimación del movimiento 6-DOF con la mejora de su anterior trabajo [29], pero con ayuda de un sensor RCB-D que lastraba las ventajas de usar las cámaras de eventos, por culpa de la latencia del sensor RGB-D, que es superior a la del sensor DVS. Otro trabajo usó imágenes para detectar las características robustas y a partir de ahí poder estimar la posición de la cámara a través de los eventos [30]. Por último, la investigación del trabajo [31] consiguió la estimación del movimiento 6-DOF además de profundidad e intensidad, con solamente la información obtenida de los sensores DVS. El problema es que estas estimaciones requieren de la reconstrucción de la intensidad, lo cual lleva a este método a ser computacionalmente muy costoso necesitando del uso de una GPU para poder llevar a cabo la estimación en tiempo real.

## EVO

Tras estas distintas investigaciones, surgió la aproximación más eficiente en el método EVO (Event-based Visual Odometry) [32]. Este mejoraba al resto de sistemas desarrollados hasta el momento gracias a que es puramente geométrico, lo cual descarta el uso de la estimación de la intensidad de la imagen, por lo que previene el error acumulado de dicha estimación, y es computacionalmente eficiente ejecutándose en una CPU a tiempo real.

EVO sigue el principio de separación en dos módulos de los métodos SLAM, uno para la estimación del movimiento y otro para el mapeado que van retroalimentándose entre sí. La posición se nutre del mapa semidenso en tres dimensiones de la escena usando los eventos, y el mapa semidenso se va expandiendo cuando se van encontrando nuevos eventos, apoyándose en la posición de la cámara.

El módulo de seguimiento de la cámara se basa en el principio de imagen-modelo que siguen otros sistemas SLAM para imágenes estándar, este consiste en alinear con el mínimo error posible los puntos de la imagen con el mapa 3D ya establecido. Pero en este método se cambia el error fotométrico, que consiste en la diferencia de intensidad en los píxeles de dos fotogramas que enfoquen el mismo punto del entorno, por el error del alineamiento geométrico entre dos imágenes con bordes de los objetos (donde ocurren los eventos normalmente).

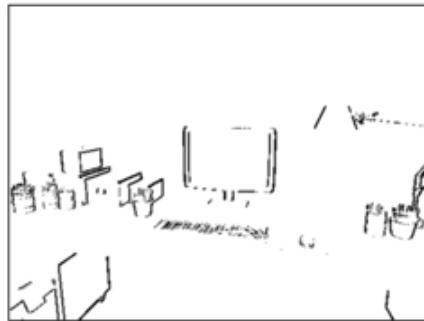


Figura 1.5: Imagen conseguida con el sensor



Figura 1.6: Proyección del mapa 3D semidenso

Dada la naturaleza de las imágenes obtenidas por los sensores DVS donde se muestran bordes, con la función inversa a la de Lucas-Kanade [33], un algoritmo iterativo de alineamiento para imágenes que permite alinear ambas imágenes, la conseguida con el sensor 1.5, y la obtenida con la pro-

yección del mapa 3D [1.6], con el objetivo de estimar las posiciones de la cámara. La inversa de la función Lucas-Kanade permite una carga computacional baja con respecto a las otras variaciones de la función, ya que las derivadas de la proyección del mapa 3D pueden ser precalculadas puesto que permanece constante durante la iteración. Y finalmente se le pasa un filtro de mediana para obtener una trayectoria más suave de la cámara de eventos.

Se sigue pues un enfoque basado en fotogramas clave, donde se seleccionan las posiciones con respecto al movimiento del sensor y un nuevo mapa local de la escena se construye a partir de la creación de este fotograma clave. Por lo tanto se debe crear un fotograma clave cada vez que el mapa 3D sea insuficiente para el seguimiento de la cámara, para ampliarlo. El sistema crea un fotograma clave siempre que la distancia entre la posición actual de la cámara y el último fotograma clave, dividida por la profundidad media de la escena, alcance un umbral (por ejemplo, el 15 %). Esto garantiza que el mapa se actualice regularmente a medida que el sensor DVS se mueve.

El módulo de mapeado se encarga de resetear el mapa 3D creando uno nuevo a partir de los últimos 2 millones de eventos, mientras está en proceso el módulo de seguimiento seguirá utilizando el anterior mapa y lo reemplazará tan rápido como sea posible. En caso de no necesitar un nuevo mapa, este se va actualizando cada 100.000 eventos nuevos refinando la nube de puntos ya existente con la nueva información extraída.

## SVO

Como a la hora de desarrollar el proyecto no estaba disponible en código abierto y las restricciones sanitarias imposibilitaban la recogida y uso de las retinas artificiales se han tenido que buscar acercamientos tradicionales al problema de estimación de la posición y la velocidad 3D de la cámara a través de fotogramas clásicos. La mayoría de algoritmos para la odometría visual se basan en PTAM [34], es un algoritmo de SLAM basado en características robustas captadas en la escena que logra la robustez mediante el seguimiento y el mapeado de cientos de estas características, como podemos observar por ejemplo en la Figura [1.7], estas características se sitúan en los píxeles donde haya un cambio de intensidad con respecto a los píxeles cercanos.



Figura 1.7: Ejemplo extracción características robustas

Al mismo tiempo, se ejecuta en tiempo real paralelizando las tareas de estimación de movimiento y mapeo como ya hemos visto anteriormente. Sin embargo, PTAM se diseñó para aplicaciones de realidad aumentada en pequeñas escenas de escritorio y fueron necesarias múltiples modificaciones (por ejemplo, limitar el número de fotogramas clave) para permitir el funcionamiento en entornos exteriores a gran escala.

Una de las propuestas que usaré en el proyecto es SVO (Semi-Direct Visual Odometry) [35], que basa su funcionamiento en las correspondencias de características entre imágenes, pero esta es calculada con una estimación directa del movimiento y no de los métodos convencionales de extracción de características y correlación. Así, la extracción de características sólo es necesaria cuando se selecciona un fotograma clave para inicializar nuevos puntos 3D. La ventaja es una mayor velocidad debido a la ausencia de extracción en cada fotograma y una mayor precisión gracias a la correspondencia de características a nivel de píxeles.

A diferencia de los métodos de odometría visual directa, se utilizan cientos de cuadros delimitadores pequeños en lugar de unas decenas de cuadros grandes. Sin embargo, demuestra que la escasa información de profundidad es suficiente para obtener una estimación aproximada del movimiento y encontrar las correspondencias de las características. En cuanto se establecen

las correspondencias de características y una estimación inicial de la postura de la cámara, el algoritmo continúa utilizando sólo características puntuales, de ahí el nombre de “semi-directo”.

Como en EVO, el sistema SVO se divide en dos módulos paralelos, uno para la estimación del movimiento de la cámara y otro para el mapeado del entorno explorado. Consiguiendo así separar ambos módulos y separando el mapeado de la restricción de trabajarse en tiempo real.

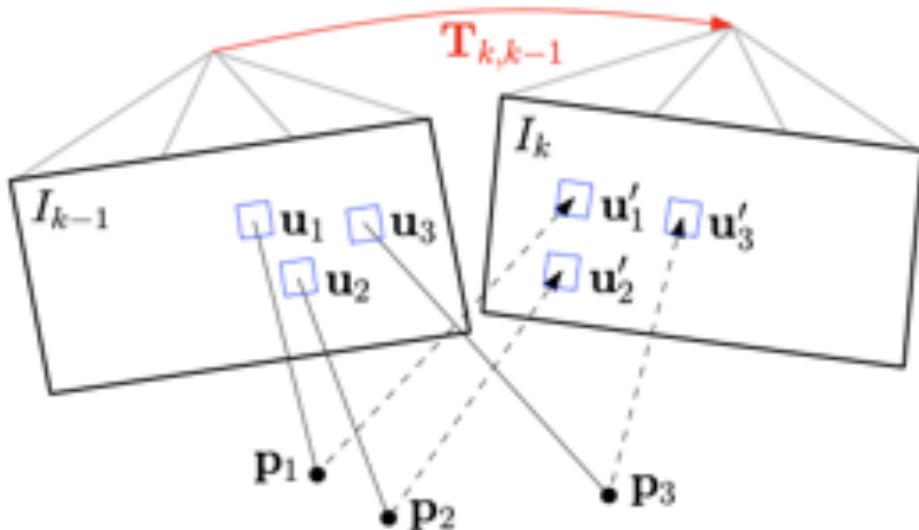


Figura 1.8: Cambiando la posición relativa  $T_{k,k-1}$  entre el fotograma anterior ( $k-1$ ) y el nuevo ( $k$ ), mueve implícitamente la proyección de puntos en el espacio para el nuevo fotograma. El modelo disperso para la alineación de la imagen pretende minimizar el error cometido en los cuadrados azules entre fotogramas que pertenezcan a los mismos puntos del espacio

El primer paso es la inicialización de la posición a través de la alineación de la imagen basada en un modelo disperso, la posición de la cámara en relación con el fotograma anterior se obtiene minimizando el error fotométrico entre los píxeles correspondientes a la ubicación proyectada de los mismos puntos 3D como se ve en la Figura 1.8. Después se pasa por unos filtros que corrigen cada cuadrado individualmente y se vuelve a optimizar en todo el fotograma para reducir el error cometido por ese paso.

En el módulo de mapeado, se inicializa un filtro de estimación de profundidad por cada punto en 2D encontrado en el que se estima su punto en el entorno 3D. Por cada vez que las correspondencias entre estos puntos

2D a los puntos en el entorno 3D son insuficientes, se genera un nuevo filtro de estimación de profundidad. Los filtros se inicializan con una gran incertidumbre en la profundidad, así que en cada fotograma posterior, la estimación de la profundidad se actualiza de forma bayesiana. Cuando la incertidumbre de un filtro de profundidad es lo suficientemente pequeña, se inserta un nuevo punto 3D en el mapa y se utiliza inmediatamente para la estimación del movimiento, retroalimentándose ambos procesos tal y como después harían en EVO.

En el artículo de EVO [32] se encuentra una comparativa entre ambos métodos usando una propia reconstrucción en escala de grises proporcionada por EVO del conjunto de datos de cámaras de eventos [17].

### 1.2.5. Algoritmos para la reconstrucción 3D y estimación de profundidad a través de eventos

Finalmente revisamos las técnicas desarrolladas para la reconstrucción 3D a través de la información de los eventos. La mayoría de trabajos para estimar la profundidad y así llegar al objetivo se han realizado usando dos o más sensores DVS simultáneamente, colocados en una misma base y compartiendo el mismo temporizador. Para poder llevarlo a cabo, primero se resuelve la correspondencia de los eventos de cada cámara para después triangular y situar el punto en tres dimensiones. Existen dos aproximaciones para el primero de los pasos, la primera de ellas se hace con técnicas tradicionales en visión estéreo en fotogramas artificiales pero acumulando los eventos en el tiempo [36], o explotando las correlaciones temporales y simultaneidad de los eventos a través de las cámaras [37].

Pero con lo costosos que son los sensores DVS, se implora una solución monocular para la reconstrucción 3D. En la sección anterior 1.2.4 se comentó el trabajo [31], que además de resolver el problema del seguimiento del sensor, era capaz de mostrar una representación 3D del entorno, pero como ya vimos necesitaba de mucha potencia hardware haciendo uso de una unidad GPU por la necesidad de estimar profundidad e intensidad de la imagen para llevarlo a cabo. El método EMVS: Event-Based Multi-View Stereo [38] aprovecha directamente la dispersión del conjunto de eventos para la reconstrucción 3D, sin hacer uso de la intensidad de la imagen para estimar la profundidad como el anterior método, dando lugar a una ejecución eficiente siendo posible ejecutarlo a tiempo real con una CPU.

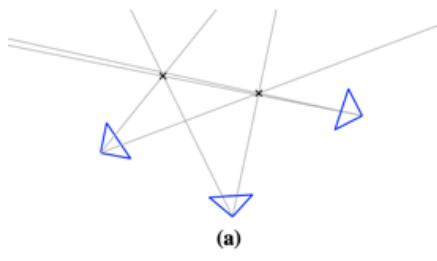
A pesar de no poder usar los mismos métodos clásicos para el problema con MVS (Multi-View Stereo) ya que se necesitan las intensidades de la imagen, sí que se basan en su trabajo para crear un nuevo método capaz de

resolver el espacio del entorno [39]. Este método consiste en dos pasos, dar una puntuación de consistencia agregada a un volumen de interés discretizado, que será la imagen del espacio de disparidad (DSI) y, a continuación, encontrar información de la estructura 3D en este DSI.

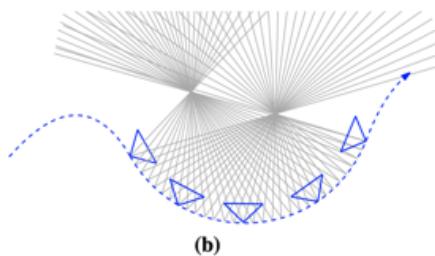
Dos diferencias a tener en cuenta entre los métodos son que este DSI, al usar información de eventos habrá huecos en algunos voxels [40], el equivalente a los píxeles en tres dimensiones, a diferencia del DSI formado por intensidad de la imagen que siempre habrá valor en un voxel, y en el MVS se reconocen los objetos de la escena gracias al DSI, pero en EMVS es mejor encontrar estructuras semidensas como curvas o puntos debido a la escasez de información en el DSI.

Por lo tanto los pasos que sigue el algoritmo EMVS son los siguientes:

**1º Rayos de puntos característicos por retroproyección de eventos:** Comienza usando la información de los eventos obtenidos por el sensor para obtener unos rayos propagados que van desde estos eventos localizados en una posición hasta la posición de la cámara de eventos. Esta posición de cada evento viene dada por  $(x,y)$  que indican el ancho y alto del evento captado por el sensor, además se almacena una marca de tiempo  $t$  con la que podremos asociar la posición del sensor en el espacio, y la polaridad  $p$ , que indica el signo del cambio del brillo.



(a)



(b)

Figura 1.9: Rayos propagados con MVS usando fotogramas clásicos, dependiendo de la velocidad de la cámara.

Figura 1.10: Rayos propagados con eventos, disparados cada vez que se mueve el sensor.

Como observamos en la Figura 1.9, esta mayor abundancia de mediciones y puntos de vista en la configuración basada en eventos genera muchos más rayos de visión que en el MVS basado en fotogramas, y por lo tanto, facilita la detección de puntos de la escena 3D mediante el análisis de las regiones de alta densidad de rayos. En la Figura 1.10 podemos ver como gracias a la gran cantidad de rayos que se obtienen EMVS no necesita asociar un evento a un punto 3D para recuperar su posición 3D, no como en

otros métodos que sí necesitan esa asociación mediante correspondencias de técnicas estéreo para poder estimar la profundidad

**2º Recuento de rayos volumétricos. Creación de la imagen del espacio de disparidad (DSI):** El segundo paso del método es discretizar el volumen que contiene la escena 3D y contar la cantidad de rayos que pasan por cada voxel usando el DSI. Para ello se crea un entorno virtual desde una de las posiciones del sensor de eventos que están asociados a un subconjunto de eventos seleccionados, definiendo así el DSI con un volumen adaptado al campo de visión del sensor DVS, como se puede observar en la Figura 1.11 existen esos voxels representados desde la posición RV en la imagen, cada voxel con su puntuación obtenida por la cantidad de rayos que pasan por el mismo, dejando en blanco los voxels donde no pasa ningún rayo, azul claro por los que pasa el rayo retroproyectado por el evento a las distintas posiciones del sensor conocida una vez, y azul oscuro cuando pasan los dos rayos por el voxel.

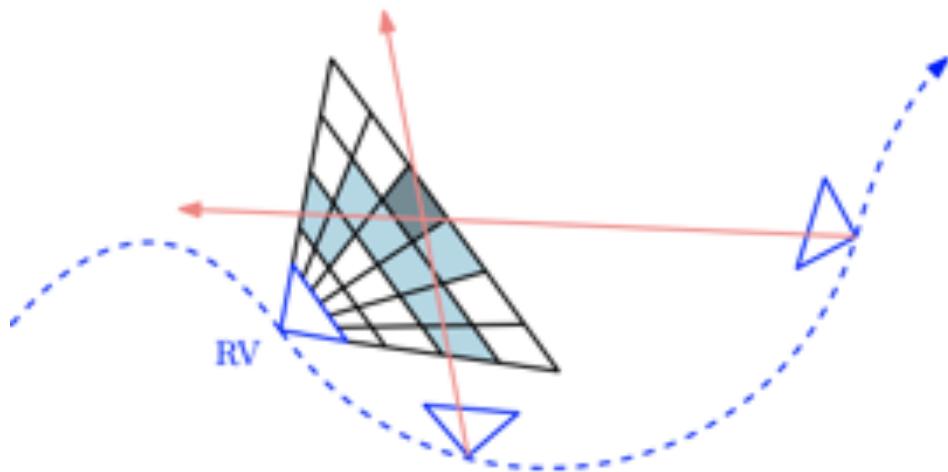


Figura 1.11: Creación del DSI contando los rayos que pasan por campo proyectado desde el punto RV.

**3º Detección de la estructura de la escena mediante la maximización de la densidad de rayos:** Se obtiene un mapa semidenso de profundidad a partir del DSI, considerando si un punto en las tres dimensiones está o no en el entorno 3D. Esa decisión se toma a partir de la función explicada en el segundo paso, donde lo normal es que los puntos de la escena se encuentren en la escena sean aquellos donde el voxel sea un máximo local de la función.

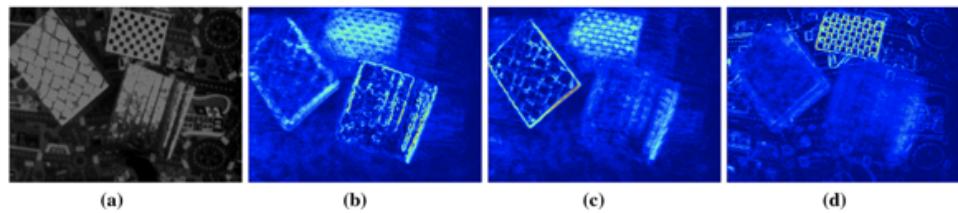


Figura 1.12: a) Imagen del entorno virtual b) DSI en profundidad corta c) DSI en profundidad media d) DSI en profundidad larga.

En la Figura 1.12 se puede ver el resultado del DSI en un entorno con tres cajas a distintas profundidades, sacadas del propio DSI generado en 3D 1.13.

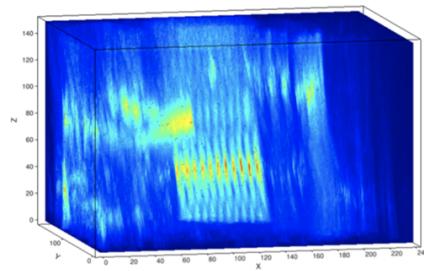


Figura 1.13: DSI con la densidad de rayos.

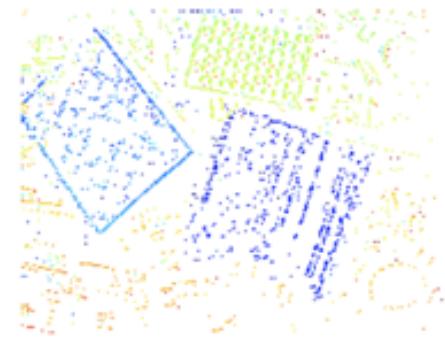


Figura 1.14: Mapa semidenso de profundidad.

**4º Fusión de mapas de profundidad desde distintos puntos de vista:** Como hemos visto en el segundo paso, se va seleccionando un distinto punto de vista en cuanto se excede en un porcentaje la media de la distancia actual, usando el mismo subconjunto de eventos hasta encontrar el punto de vista que no exceda ese porcentaje y así ir creando el mapa semidenso de profundidad 1.14. A continuación, los mapas semidensos de profundidad se convierten en nubes de puntos, eliminando los puntos aislados, aquellos cuyo número de vecinos dentro de un radio determinado es inferior a un umbral, y se fusionan en una nube de puntos global 1.15 utilizando las posiciones conocidas de los entornos virtuales.

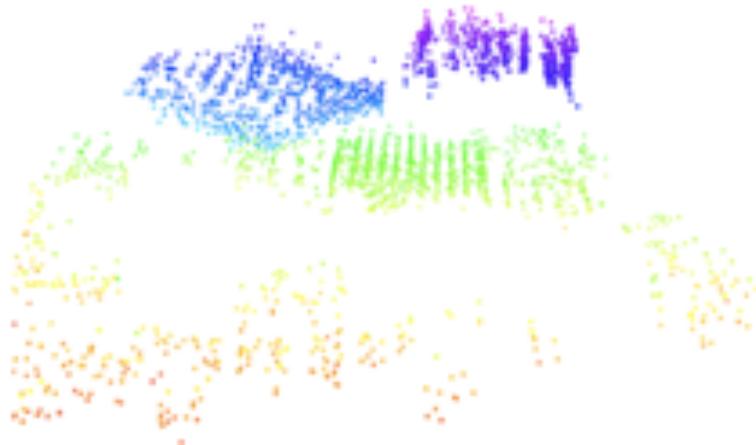


Figura 1.15: Nube de puntos.

**5º Limpieza del mapa:** Para mejorar la calidad de la reconstrucción 3D se usan filtros en los mapas semidensos de profundidad, pasándole un filtro de mediana a aquellos puntos en el espacio resultante tras aplicar otro filtro adaptativo Gaussiano. Esto permite eliminar aquellos puntos que parecen alejados a la estimación real (outliers).

Además se le aplica un filtro de radio eliminando los puntos de la nube de puntos generada que no tengan un mínimo de vecinos en el radio establecido, eliminando así puntos aislados, los cuales normalmente son outliers.

### 1.3. Objetivos

Este Trabajo Fin de Grado tiene como objetivo desarrollar un sistema capaz de realizar una reconstrucción 3D del entorno a partir de un sensor de eventos, basado en las distintas técnicas ya existentes en el estado del arte procurando cumplir los siguientes objetivos:

- Estudiar el estado del arte de los eventos y los métodos para su simulación, técnicas para la reconstrucción 3D y para la estimación de la posición de la cámara.
- Construir un sistema capaz de realizar una reconstrucción de un entorno en tres dimensiones a partir de datos de un sensor de eventos.
- Estudiar las ventajas de usar un simulador de eventos.
- Estudiar las funcionalidades y herramientas que ofrece el middleware ROS.

- Desarrollar un paquete software para integrar los diferentes módulos del sistema.
- Realizar un sistema en código abierto utilizando herramientas de software libre disponible para cualquier usuario que lo requiera. [41]
- Desarrollar un sistema extensible, capaz de adoptar de forma sencilla nuevas funcionalidades al sistema.

## 1.4. Estructura del proyecto

Se presentan los diferentes capítulos que conforman la memoria de todo el desarrollo del proyecto, acompañados de una breve descripción.

### CAPITULO 1: Introducción

Contextualización del problema que se pretende resolver junto a la motivación para la elección del proyecto. Además se relatan detalladamente la información relativa a los sensores de eventos, junto a los distintos campos necesarios del proyecto.

### CAPITULO 2: Gestión del proyecto

En este capítulo se muestra la metodología seguida para la realización del proyecto, junto a la planificación temporal representada con un diagrama de Gantt que detalla el tiempo dedicado a cada una de las fases. Finalmente se expone el presupuesto con todos los costes requeridos.

### CAPITULO 3: Entorno Operacional

Exposición de las distintas herramientas Hardware y Software que se emplean en el trabajo, con las especificaciones de los distintos dispositivos usados para el proyecto, junto a una explicación a fondo de los conceptos de la herramienta ROS enfocada al problema a abordar en el trabajo.

### CAPITULO 4: Diseño

Se muestra un esquema del sistema al completo junto a una explicación de la función de cada uno de sus componentes, además de lo que requieren unos de otros, para así dar paso a los casos de uso y los requisitos funcionales y no funcionales. Finalmente, se encuentra un diagrama de flujo del sistema para una mejor comprensión a la hora de utilizar el sistema.

### CAPITULO 5: Implementación

Este capítulo detalla todas las modificaciones y creaciones de código realizadas para la correcta integración de los sistemas en el proyecto. Además, concluye el capítulo con el manual de uso del proyecto, también disponible en Github.

**CAPITULO 6: Problemas**

Se describen las distintas restricciones para llevar a cabo el proyecto, junto a los problemas técnicos e imprevistos que han ido sucediendo durante la realización del trabajo.

**CAPITULO 7: Experimentos**

Explicación de las distintas pruebas realizadas para la evaluación de cada componente del proyecto, junto a un análisis de resultados en cada una de las pruebas apoyado en imágenes, gráficas y tablas, además de una discusión final en los resultados obtenidos por las distintas pruebas con unas pequeñas conclusiones.

**CAPITULO 8: Conclusiones y trabajo futuro**

Se obtienen las conclusiones finales de los resultados obtenidos en el proyecto junto a un repaso de los objetivos cumplidos, y finalmente se exponen posibles mejoras aplicables al sistema en un trabajo futuro.

**Bibliografía**

Enumera en una lista las distintas referencias de publicaciones seguidas a lo largo de la documentación de la memoria.

## Capítulo 2

# Gestión del proyecto

### 2.1. Metodología

He utilizado la metodología de Desarrollo de prototipos [42] debido a que nuestro proyecto se puede dividir en los distintos sistemas/módulos necesarios para el correcto funcionamiento de todo el trabajo, de forma que podemos ir probándolos y refinándolos uno a uno de cara a optimizar el resultado final [2.1].

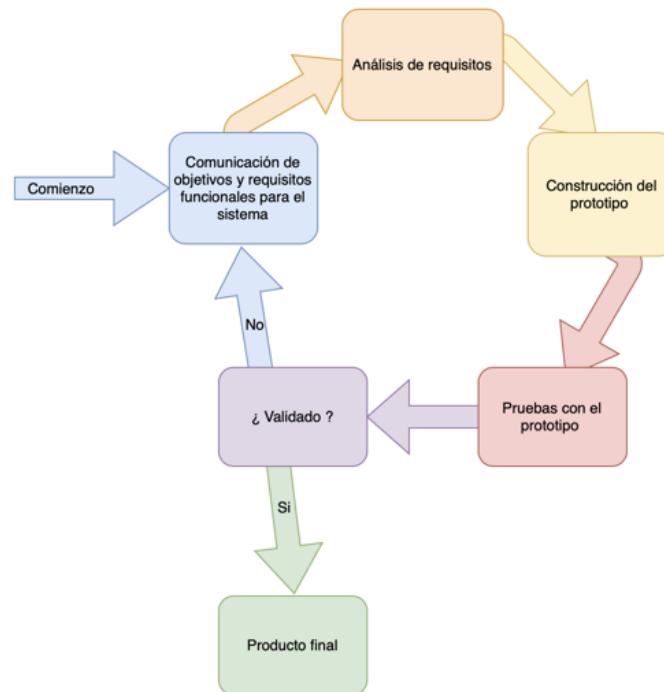


Figura 2.1: Esquema del desarrollo de prototipos

Debido a la naturaleza del proyecto, esta metodología es idónea ya que sabemos que el objetivo general es conseguir una reconstrucción 3D con los datos extraídos de una cámara de eventos, pero no sabemos los requisitos detallados de entrada, procesamiento o salida. Además, la facilidad que nos otorga el middleware ROS para acoplar los distintos sistemas y así tener una mejor idea de la eficacia de cada uno de ellos en nuestra máquina, ya que no tenemos la seguridad en el rendimiento que aportaran todos estos sistemas en nuestro trabajo. Por ello este es el modelo de desarrollo que he elegido conforme a los motivos dados y que vendrá muy bien para saber el progreso poco a poco sin tener que esperar al final del proyecto.

## 2.2. Planificación temporal

A lo largo del proyecto se han ido recopilando las fechas en las que se trabajaban cada una de las fases, siempre intentando seguir la planificación ideada por el tutor al comienzo del cuatrimestre pero que su cumplimiento fue complicado de seguir desde el principio debido a la serie de problemas que he ido encontrando por el camino. A continuación se muestra el diagrama de Gantt con el tiempo real dedicado a cada una de las fases generales y trabajo específicos [2.2]

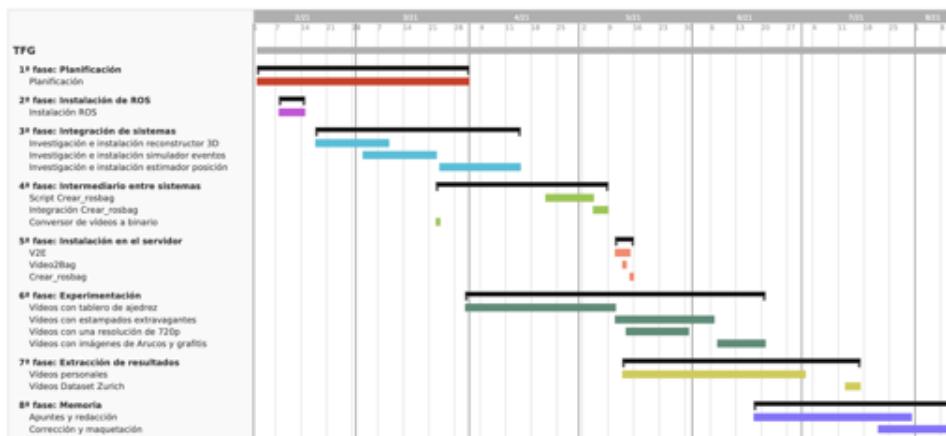


Figura 2.2: Diagrama de Gantt

**Fase 1: Planificación** En esta primera fase se comienza a plantear el problema a resolver, en la que se barajan las distintas opciones a abordar el trabajo con la búsqueda del tema principal, y una vez escogido, se fija una idea general y los distintos puntos del sistema. Estudiando y analizando intensivamente el estado del arte de las distintas funcionalidades requeridas para el proyecto. Para esta fase se planificó originalmente un tiempo esti-

mado de 6 semanas.

**Fase 2: Instalación de ROS** Una vez establecido el tema principal, se busca el entorno de trabajo en el que se desarrollará el sistema y la instalación del mismo. Para esta fase se planificó originalmente un tiempo estimado de 1 semana.

**Fase 3: Integración de sistemas** En esta fase se integran las distintas alternativas escogidas tras su profundo estudio, incorporándolas al proyecto las disponibles en el middleware ROS, o por el contrario en un entorno de anaconda. Esta finaliza una vez se consigue integrar los sistemas principales, comprobado su funcionamiento con los ejemplos proporcionados por los autores. Para esta fase se planificó originalmente un tiempo estimado de 6 semanas.

**Fase 4: Intermediarios entre sistemas** Al integrar todos los sistemas necesarios para nuestro proyecto, hay que conectar la información de los distintos sistemas para un correcto funcionamiento. Por ello, esta cuarta fase consiste en el desarrollo de distintos scripts capaces de unir la información de salida en un formato adaptado para la correcta comprensión del siguiente módulo del sistema. Para esta fase se planificó originalmente un tiempo estimado de 4 semanas.

**Fase 5: Instalación en el servidor** Para esta quinta fase se lleva parte del proyecto a un servidor prestado por la UGR, dotado con una tarjeta gráfica NVIDIA capaz de mejorar la eficiencia en tiempos para distintos módulos del sistema, como el simulador de eventos V2E, además de otorgar más flexibilidad en los parámetros seleccionables, permitiendo una mayor variedad en las pruebas. Para esta fase se planificó originalmente un tiempo estimado de 1 semana.

**Fase 6: Experimentación** La sexta fase del proyecto consiste en las distintas pruebas que se realizan para la comprobación de la viabilidad del sistema. Siguiendo un rango distinto de parámetros y vídeos en entornos distintos para buscar los mejores resultados. Para esta fase se planificó originalmente un tiempo estimado de 8 semanas.

**Fase 7: Extracción de resultados** Una vez se tienen cada una de las pruebas distintas, se procede a analizarlas y compararlas entre sí, además con los resultados del dataset proporcionado por la Universidad de Zurich. Para esta fase se planificó originalmente un tiempo estimado de 4 semanas.

**Fase 8: Memoria** En esta última fase se realizan los distintos capítulos de la memoria y la posterior revisión de los mismos, donde se exponen los distintos aspectos del trabajo Fin de Grado. Para esta fase se planificó originalmente un tiempo estimado de 5 semanas.

Por lo general cada fase ha necesitado más tiempo del esperado en un principio, sobretodo al instalar los distintos módulos que he ido necesitando para el proyecto, inconvenientes que se profundizarán en el capítulo de problemas, han provocado continuos retrasos llegando a la ampliación de tiempo requerido para la realización del proyecto entre unas 5/6 semanas más de las inicialmente planificadas.

La situación extraordinaria en la que nos encontramos con la pandemia del COVID-19, llevando el proyecto a un distinto enfoque por la falta de material y la necesidad de integrar varios sistemas para compensarlo. Sumado a la necesidad de refinar los datos de entrada para optimizar los resultados obtenidos por el reconstructor 3D, han requerido de una mayor cantidad de tiempo en la fase de experimentación y por consiguiente un retraso en la fase de extracción de resultados.

### 2.3. Hitos

Estos objetivos parciales han ido variando mucho a lo largo del proyecto debido a los distintos problemas que han ido surgiendo debido a las limitaciones sufridas que se explicarán más a fondo en un posterior capítulo de problemas, detallando los distintos acercamientos que finalmente se descartaron y dieron lugar a los que finalmente han quedado en el trabajo final.

Instalación ROS Noetic -> Inicio: 1<sup>a</sup> semana del proyecto

Distribución de ROS compatible con el SO disponible en mi maquina personal Ubuntu 20.04.

Selección del método de reconstrucción 3D/SLAM -> Inicio: 2<sup>a</sup> semana del proyecto

Repaso de las distintas técnicas disponibles e instalación en el proyecto.

Selección método e instalación de un simulador de eventos a partir de vídeos -> Inicio: 4<sup>a</sup> semana del proyecto

Buscar un sistema funcional que convierta vídeos a eventos y sea compatible para nuestro proyecto.

Método para sacar posiciones de la cámara e instalación SVO -> Inicio: 7<sup>a</sup> semana del proyecto

Instalación del método recomendado por los autores del reconstructor 3D para situar la cámara en los vídeos.

Grabación vídeos de prueba -> Inicio: 8<sup>a</sup> semana del proyecto

Vídeos personales para generar eventos con el simulador y probar el estimador de la posición de la cámara, probando distintas configuraciones y movimientos para optimizar los resultados obtenidos.

Creación script para juntar datos obtenidos -> Inicio: 11<sup>a</sup> semana del proyecto

Código para unir los datos obtenidos por el simulador de eventos y las posiciones de la cámara para el funcionamiento del reconstructor 3D con vídeos personales.

Extracción resultados 3D con vídeos personales -> Inicio: 13<sup>a</sup> semana del proyecto

Obtención y optimización de los resultados obtenidos por los vídeos personales, cambiando los distintos parámetros e instantes de tiempo.

Instalación proyecto en servidor -> Inicio: 14<sup>a</sup> semana del proyecto

Uso del servidor con una GPU Nvidia RTX 2080 Ti con capacidad de usar CUDA ( Compute Unified Device Architecture ) [43], consiguiendo tiempos de ejecución más bajos en el simulador de eventos y los distintos scripts.

## 2.4. Presupuesto

El presupuesto de este proyecto se ha calculado de la siguiente manera:

$$\frac{D}{V} * C \quad (2.1)$$

Donde:

- D es el número de meses que se ha utilizado el producto para el proyecto.
- V es la vida útil del producto en número de meses.
- C es el coste total del producto en euros.

#### 2.4.1. Recursos Hardware

Articulo	Dedicación (meses)	Vida Útil (meses)	Coste (euros)	Coste aplicable al proyecto (euros)
Ordenador personal	6	60	2500	250
Servidor con Nvidia RTX 2080 Ti	3	120	2800	70
<b>Total</b>				<b>320</b>

Cuadro 2.1: Coste en recursos Hardware

#### 2.4.2. Recursos Software

Articulo	Dedicación (meses)	Vida Útil (meses)	Coste (euros)	Coste aplicable al proyecto (euros)
Ubuntu 20.04	6	-	0	0
Parallels Virtual Machine	6	12	80	40
ROS Noetic	6	-	0	0
Anaconda	6	-	0	0
Visual Studio Code	6	-	0	0
<b>Total</b>				<b>40</b>

Cuadro 2.2: Coste en recursos Software

#### 2.4.3. Recursos Humanos

Para la realización del proyecto se han requerido 6 meses de trabajo, son aproximadamente unos 130 días, teniendo en cuenta que no todos los días de la semana se ha trabajado en el proyecto, y cada día se emplearon 4 horas de media. Según [44], un ingeniero informático recién egresado tiene un sueldo anual que ronda entre los 18.000 y 22.000 euros. Esto supone un gasto de cercano en media de 1700 euros al mes, sumando los costes extra requeridos por parte de la empresa, como la seguridad social, se queda en unos 2700 euros al mes en un trabajador con jornada completa, o lo que es lo mismo, 8 horas diarias y 5 días a la semana. Se queda en 13.5 euros la hora. Considerando las 520 horas dedicadas al proyecto, se necesitaría un presupuesto de 7020 euros.

Recurso	Coste (euros)
<b>Hardware</b>	320
<b>Software</b>	40
<b>Humanos</b>	7020
<b>Total</b>	7380
<b>Total + amortiguador financiero (10 %)</b>	8118

Cuadro 2.3: Coste total para el proyecto

En definitiva se requieren 8118 euros en total para llevar a cabo un proyecto como el que se describe en este trabajo fin de grado.

## Capítulo 3

# Entorno Operacional

En este capítulo se expondrán tanto las distintas herramientas software, como los dispositivos hardware usados para la realización del proyecto. Este estará compuesto por diferentes sistemas, por lo que nos podremos aprovechar de la organización que ofrece ROS, módulos con distintas funcionalidades fácilmente integrables, comunicándolos entre sí para obtener la solución de nuestro problema. También, nos serán útiles las librerías disponibles en ROS para el tratamiento de la información de los sensores DVS, siendo el mejor entorno disponible para el enfoque de nuestro trabajo.

Además se detallan a fondo las características de los distintos dispositivos hardware usados para la captación de los vídeos como para la ejecución del sistema.

### 3.1. ROS

Es un middleware libre desarrollado para facilitar la construcción de aplicaciones para robots como bien indica su nombre Robotic Operating System [11]. Surgió por la necesidad de estandarizar la programación del comportamiento de los robots, con ayuda de una gran cantidad de bibliotecas y herramientas.

Siendo software libre, se le pueden añadir contribuciones de otros usuarios, que resuelvan problemas más generales, y así ayudarse para resolver el problema más específico, todo gracias a la organización en paquetes independientes de la que dispone ROS.

Para la comunicación entre los distintos paquetes ROS se basa en un paradigma distribuido de publicadores-subscriptores, donde la información se

transmite a través de un nodo publicador y otro nodo subscriptor recibe la información.

Para ello ROS se basa en una serie de conceptos para su funcionamiento:

- Paquetes: Es la unidad independiente en la que se basa ROS en la que, idealmente, se almacena una funcionalidad específica para poder utilizarlo en distintos proyectos. Esta funcionalidad puede ir determinada por nodos, una biblioteca independiente, archivos de configuración, un conjunto de datos, un software de terceros o cualquier compuesto de archivos que forme un módulo útil.
- Nodos: Son los procesos de ROS que se encargan de comunicar la información usando librerías específicas de ROS, tanto para publicarla a otros como para recibir la información.
- Topics: Estos son los canales de comunicación donde los nodos pueden leer (suscriptor) y escribir (publicador) mensajes. Puede haber varios nodos suscriptores o publicadores, pero únicamente se saben los topics que están siendo publicados y se desconocen los nodos que publican o se suscriben a la información.
- Servidores de parámetros: Es un diccionario compartido al que acceden los distintos paquetes de ROS. En él se almacenan los parámetros en tiempo de ejecución donde los nodos pueden visualizarlos para inspeccionar de forma sencilla el estado de configuración del sistema y modificarlo si se necesita. Está diseñado para guardar los datos estáticos al no estar preparado para un alto rendimiento, como por ejemplo los datos de configuración del sistema.
- Launchfiles: Estos son los archivos de ejecución programados en XML propios de ROS, que posibilitan ejecutar a la vez distintos nodos del sistema al igual que establecer parámetros en el servidor de parámetros. Se pueden ejecutar varios launchfiles a la vez.
- Mensajes: La estructura de datos con la que se comunican los nodos son los mensajes. Hay una gran variedad de tipos de mensaje pero para este trabajo solamente se necesitan “sensor\_msgs/CameraInfo.msg” [3.1](#), “dvs\_msgs/EventArray.msg” [3.2](#) y “geometry\_msgs/PoseStamped.msg” [3.3](#).

A continuación, se describen más a fondo los distintos mensajes de ROS usados en el proyecto

- CameraInfo [45]:



Figura 3.1: Estructura del mensaje CameraInfo

- Header: Cabecera con formato de un mensaje estándar de ROS que contiene un identificador, un contador que indica la posición en la secuencia de los mensajes y una marca de tiempo.
- Height y Width: La resolución de la imagen que reproduce la cámara
- Distortion\_model: El modelo de distorsión usado para la cámara, en general “plumb\_bob” es suficiente para la mayoría de cámaras.
- Matrices K, D y P: La matriz intrínseca de la cámara, parámetros del modelo de distorsión y la matriz de proyección respectivamente, para los cálculos de las matrices K y P se forman con los valores de la distancia focal de la cámara y las coordenadas del píxel central de la cámara.

- EventArray [46] [47]:



Figura 3.2: Estructura del mensaje EventArray

- Header: Al igual que con la cabecera del CameraInfo se tiene el identificador y el tiempo del frame en el que se producen los eventos.
- Height y Width: La resolución de la imagen que reproduce la cámara.
- Events: Array de eventos que son otro mensaje específico de las retinas artificiales, que contienen las coordenadas (x,y) que indican el pixel de la cámara en el que se ha producido un evento, el tiempo en el que se ha producido este evento y si es positivo o negativo el cambio de intensidad (polaridad).

- PoseStamped [48]:

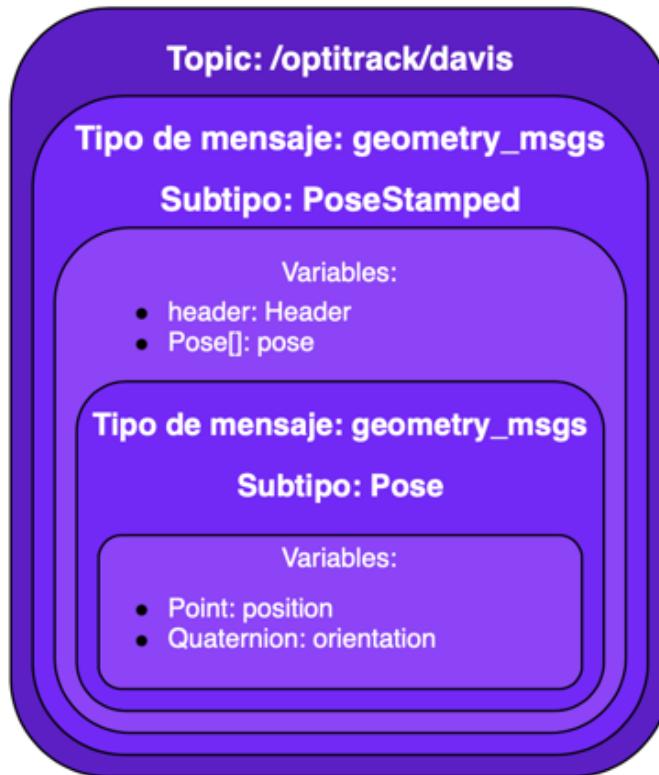


Figura 3.3: Estructura del mensaje PoseStamped

- Header: También dispone de cabecera como los otros dos mensajes.
- Position: Es un mensaje de tipo punto de la biblioteca geometry, que consta de coordenadas (x,y,z) para representar la posición en el espacio de la cámara en el tiempo indicado en la cabecera.
- Orientation: Un mensaje de tipo Quaternion de la biblioteca geometry, con coordenadas (x,y,z,w) capaz de representar la orientación en la que se dispone la cámara en el instante de tiempo indicado en la cabecera.

Para el uso de ROS en nuestra maquina una vez tenemos un paquete instalado, es necesaria la compilación del código para su posterior ejecución, para ello ROS hace uso de Rosbuild y Catkin que se explicará a continuación.

- Rosbuild fue el compilador creado desde el comienzo de ROS para la compilación y construcción de su código para su ejecución, pero que con las necesidades de la comunidad con el paso de los años se ha visto en diseños poco óptimos y una complejidad alta dio lugar a la obligación de la creación de otro compilador mejor.
- Catkin [49] es este sistema mejorado oficial de compilación para código en ROS, con una mayor compatibilidad y mejor distribución de paquetes que con su anterior compilador, obtiene los archivos Cmake de C++ y los archivos de python combinándolos dando un correcto funcionamiento al usar estos dos distintos lenguajes de programación. Gracias a las herramientas de catkin, se simplifica mucho el proceso de compilación y ejecución de código de ROS, dando lugar a su uso en distintos sistemas operativos como Microsoft Windows, donde presentaba problemas con rosbuild.
- Otra ventaja de este compilador es la poca adhesión a ROS que tiene, permitiendo usar nuestro código independiente a ROS con este sistema de compilación de código como he usado en mis propios scripts.
- Cada espacio de trabajo catkin requiere de un archivo package.xml que se encarga de:
  - Indicar el orden de la secuencia de paquetes que tienen que compilarse.
  - Definir las dependencias de cada paquete.
  - Nombre, versión, descripción y usuario que mantiene el paquete.
- Y un archivo CMakeLists.txt que se encarga de preparar y ejecutar la compilación de nuestro paquete, dando la información necesaria como la versión mínima requerida del cmake y la comprobación de la ya compilación de paquetes necesarios para la correcta compilación del paquete deseado.
- Una gran ventaja de este sistema es el aislamiento de los distintos paquetes en distintos espacios de trabajo que facilitan esa labor de modularización que otorga ROS para la contribución del código y su posterior uso en distintos proyectos si así se requieren.

Comandos importantes para el funcionamiento de ROS:

- Source: Esencial para establecer la terminal en el entorno de ROS y poder usar sus comandos.
- Roscore: Necesario para inicial el sistema ROS, inicializando su núcleo y permitiendo la comunicación entre nodos, paquetes, parámetros, etc.
- Rosrun: Ejecuta el archivo ejecutable de un paquete de ROS.
- Roslaunch: Posibilita arrancar varios nodos de ROS a la vez gracias a un archivo con formato .launch, facilitando así el uso de sistemas diferentes ejecutándolos simultáneamente.
- Rosls: Muestra el contenido del paquete en el que estamos situados en terminal.
- Roscd: Accede al directorio del paquete de ROS que haya sido compilado.
- Rospack: Muestra la información de un paquete.
- Rosnode: Puede mostrar la lista de nodos en ejecución, la información de uno de ellos o la posibilidad de parar la ejecución de alguno de ellos.
- Rostopic: Entre sus funciones podemos mostrar la lista de topics que se están publicando en ese instante, obtener información de alguno de ellos o leer la información de un topic específico.

### 3.2. Ordenador propio y SO (Ubuntu 20.04)

Todo el proceso de instalación y desarrollo del trabajo se ha realizado a través de un ordenador portátil Macbook Pro con las siguientes características:

- CPU: Intel Core i7 de 6 núcleos con 2.2 GHz de velocidad
- GPU: AMD Radeon Pro 555X
- RAM: 16 GB DDR4

Al usar mi ordenador personal portátiluento con la facilidad que eso supone a la hora de poder continuar mi trabajo en los distintos lugares a los que me he tenido que desplazar, al igual que la rapidez de mostrar mis dudas y resultados al tutor en su despacho cuando las medidas sanitarias lo han permitido. Pero en uno de los simuladores haría uso de CUDA para aprovechar la potencia de la GPU y reducir drásticamente el tiempo que tardaría

en cada ejecución del simulador, por ello también he tenido que hacer uso de un servidor que tenga una tarjeta NVIDIA que se especificará en el 3.3.

Existen versiones de ROS para Mac OS pero son experimentales, por ello me decanté por usar Ubuntu 20.04, ya que cuenta con una versión estable de ROS, ROS Noetic. Con el uso de la máquina virtual Parallels, la última versión de la distribución de Linux es la que más fluida y mejor rendimiento aporta, además de ser compatible con ROS Noetic, última versión estable disponible, y que, a pesar de tener la mayoría de paquetes a usar compatibles con ROS Kinetic, esta versión se ha dejado de dar soporte oficial en el presente año 2021. Por ello la elección de ROS Noetic en Ubuntu 20.04 y por la gran comunidad de usuarios que se encuentran usándolo y dándole soporte.

### **3.3. Servidor UGR con tarjeta gráfica Nvidia RTX 2080 Ti**

Durante la integración de los distintos paquetes y las pruebas de ejecución realizados en ellos, en uno de ellos se podía hacer uso de la GPU para agilizar el proceso y gracias a mi tutor y la UGR pude hacer uso de un servidor localizado en la facultad que aprovecharía el uso de CUDA que mi portátil no puede hacer al tener una tarjeta gráfica incompatible con el servicio. Este servidor cuenta con:

- CPU: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz
- GPU: NVIDIA GeForce RTX 2080 Ti 11GB

Se me creó una cuenta de administrador en él para poder usarlo de forma remota desde mi portátil en cualquier lugar siempre y cuando esté conectado a internet.

### **3.4. iPhone SE (Cámara)**

Con la ausencia de la retina artificial debido a problemas con la pandemia he tenido que usar la cámara de mi móvil personal, un iPhone SE capaz de grabar videos de 4K, 1080p y 720p a 30/60 fotogramas por segundo dependiendo de la resolución, yo usaré en el trabajo videos grabados a 720p a 30 fps.

# Capítulo 4

## Diseño

El sistema integrado al completo 4.1 podemos dividirlo en las funciones o módulos que se realizan en los distintos dispositivos que confeccionan el trabajo. El primero de los dispositivos es el iPhone o en su defecto, cualquier otra herramienta capaz de grabar un vídeo y se pueda almacenar en el ordenador para su posterior procesamiento.

Es aquí donde entra el segundo de los dispositivos de los que se hace uso en el proyecto, el servidor, donde se ha destinado los sistemas cuya carga computacional y tiempos de ejecución sean muy largos (media hora o más por ejecución), además de las ventajas al usar una GPU más potente que agiliza el proceso de ejecución. Se deja así el tercer dispositivo usado en este proyecto, mi ordenador personal, para los sistemas con menor complejidad y que no consumen mucho uso de CPU.

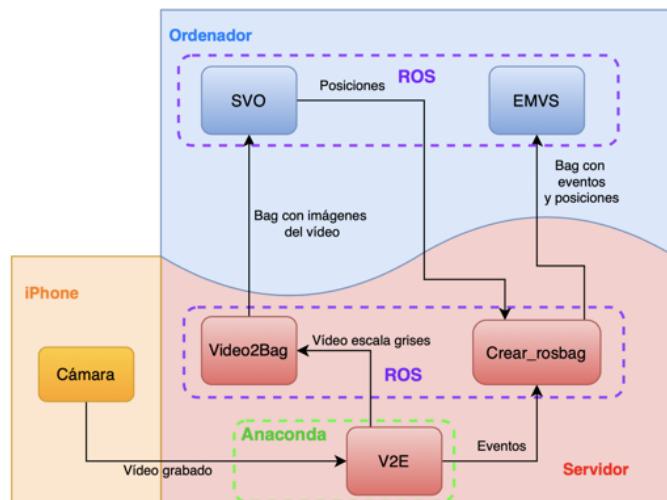


Figura 4.1: Diseño del sistema al completo

El primer módulo integrado en el servidor es el **V2E**, el simulador de eventos que como entrada recibirá el vídeo grabado por el iPhone y se encargará de generar los eventos de forma artificial. Para su uso se requiere de Anaconda, creando un entorno virtual con todas las librerías y dependencias necesarias instaladas, y la ejecución dará como salida dos archivos que se redirigen a otros sistemas que los necesitan.

Estos sistemas son el **SVO**, que requiere del video grabado en escala de grises, archivo dado por salida del V2E pero es necesario convertirlo en fichero binario para su lectura.

Usamos **Video2Bag**, otro de los módulos integrados en el servidor, como este conversor que necesita de ROS para su correcta ejecución, ya que lo convierte a un formato denominado “*bag*”, capaz de reproducirse en el entorno de ROS. Y el otro sistema que requiere de la información de salida del V2E es **Crear\_rosbag**, el último de los módulos integrados en el servidor, que recibe la información de los eventos generados por V2E para procesarla junto a las posiciones de la cámara durante el vídeo.

Estas posiciones de la cámara necesitan ser estimadas por el módulo **SVO**, que como hemos especificado antes requiere del binario formado por Video2Bag que contiene el vídeo grabado con el iPhone en escala de grises, estima las posiciones de la cámara y las publica en un *topic* de ROS, que para obtener su información, nos subscribimos desde una terminal al *topic* y la almacenamos redirigiendo la salida a un fichero de texto, para así proporcionar las estimaciones al módulo **Crear\_rosbag**.

**Crear\_rosbag** combina la información de los eventos y las posiciones de la cámara en un solo fichero binario del mismo formato que la salida del Video2Bag (*bag*), pero sin la información de los fotogramas del vídeo, sino con la información de los eventos y las posiciones de la cámara. Fichero necesario para el último de los módulos, **EMVS** dedicado a la reconstrucción 3D y estimación de profundidad a través de los eventos.

Tanto EMVS como SVO no requieren de una carga computacional alta y se pueden llevar a cabo en el ordenador personal sin que la diferencia en los tiempos de ejecución entre hacerlos desde el ordenador personal con la CPU y el servidor con ayuda de la GPU sea significativa.

## 4.1. Casos de uso

<b>CASO DE USO</b>	Generar vídeo eventos	<b>IDENTIFICADOR</b>	CU-01
<b>ACTORES</b>	Usuario		
<b>PRECONDICIONES</b>	Poseer un archivo en formato de vídeo		
<b>POSTCONDICIONES</b>	Se obtiene un vídeo con eventos simulados.		
<b>PROPSITO</b>	Obtener la simulación de eventos en un vídeo grabado por una cámara tradicional.		
<b>DESCRIPCIÓN</b>	El usuario introduce un vídeo al simulador de eventos V2E, y este le generará una simulación del mismo vídeo como si lo hubiese captado desde un sensor DVS.		

Cuadro 4.1: Caso de uso 01

<b>CASO DE USO</b>	Generar bag a partir de vídeo	<b>IDENTIFICADOR</b>	CU-02
<b>ACTORES</b>	Usuario		
<b>PRECONDICIONES</b>	Poseer un archivo en formato de vídeo		
<b>POSTCONDICIONES</b>	Se obtiene un archivo binario en formato bag que contiene los fotogramas del vídeo		
<b>PROPSITO</b>	Obtener en un fichero binario las imágenes del vídeo deseado.		
<b>DESCRIPCIÓN</b>	El usuario introduce un vídeo al conversor Video2Bag, y este le generará un archivo binario que contiene un topic con la información de cada fotograma del mismo vídeo.		

Cuadro 4.2: Caso de uso 02

<b>CASO DE USO</b>	Estimar posiciones de la cámara en un vídeo	<b>IDENTIFICADOR</b>	CU-03
<b>ACTORES</b>	Usuario		
<b>PRECONDICIONES</b>	Poseer un archivo en formato binario con las imágenes del vídeo y subscribir una terminal al topic donde se escribirán las posiciones		
<b>POSTCONDICIONES</b>	Se obtiene estimaciones de las posiciones de la cámara.		
<b>PROPSITO</b>	Obtener las posiciones de la cámara durante el vídeo		
<b>DESCRIPCIÓN</b>	El usuario reproduce un vídeo con la función de ROS rosbag play, mientras extrae las posiciones de la cámara ejecutando el estimador SVO, que publica la información de las posiciones de la cámara en un topic de ROS.		

Cuadro 4.3: Caso de uso 03

<b>CASO DE USO</b>	Generar bag a partir de información de eventos y posiciones de cámara	<b>IDENTIFICADOR</b>	CU-04
<b>ACTORES</b>	Usuario		
<b>PRECONDICIONES</b>	Poseer tres archivo en formato de texto, el primero con las posiciones de la cámara, otro con los eventos y uno último con los parámetros de calibración de la cámara.		
<b>POSTCONDICIONES</b>	Se obtiene un fichero binario que contiene información referente a los eventos, posiciones de la cámara y calibración de la misma de un mismo vídeo.		
<b>PROPOSITO</b>	Obtener un fichero con la información necesaria para el sistema EMVS.		
<b>DESCRIPCION</b>	El usuario introduce la información obtenida por los módulos V2E y SVO, además de los parámetros de calibración de la cámara que grabó el vídeo original, para así conseguir un fichero binario capaz de ejecutarse con el sistema EMVS.		

Cuadro 4.4: Caso de uso 04

<b>CASO DE USO</b>	Estimar imagen de profundidad	<b>IDENTIFICADOR</b>	CU-05
<b>ACTORES</b>	Usuario		
<b>PRECONDICIONES</b>	Poseer fichero binario con la información de un vídeo de eventos, posiciones de cámara y parámetros de calibración de la cámara.		
<b>POSTCONDICIONES</b>	Se obtiene una imagen con profundidad estimada en escala de colores.		
<b>PROPOSITO</b>	Obtener la estimación de profundidad en una escena captada a partir de eventos.		
<b>DESCRIPCION</b>	El usuario introduce un fichero binario con la información de los eventos al reconstructor 3D, y este devuelve una imagen con distintos colores, dependiendo de la profundidad estimada en los píxeles donde existían eventos.		

Cuadro 4.5: Caso de uso 05

<b>CASO DE USO</b>	Generar reconstrucción 3D	<b>IDENTIFICADOR</b>	CU-06
<b>ACTORES</b>	Usuario		
<b>PRECONDICIONES</b>	Poseer fichero binario con la información de un vídeo de eventos, posiciones de cámara y parámetros de calibración de la cámara.		
<b>POSTCONDICIONES</b>	Se obtiene una nube de puntos con la reconstrucción 3D generada.		
<b>PROPOSITO</b>	Obtener la reconstrucción 3D de una escena captada a partir de eventos.		
<b>DESCRIPCION</b>	El usuario introduce un fichero binario con la información de los eventos al reconstructor 3D, y este devuelve una nube de puntos en 3D, generada a partir de las estimaciones de profundidad calculadas.		

Cuadro 4.6: Caso de uso 06

## 4.2. Requisitos

En esta sección se describen, definen y especifican todas las características necesarias para el desarrollo del sistema, dividiendo estos requisitos en dos principales grupos. Funcionales y no funcionales.

### 4.2.1. Funcionales

Los requisitos funcionales describen de forma más detallada las funcionalidades o servicios del sistema desde el punto de vista del sistema.

- **RF01:** El sistema permitirá simular eventos a partir de un archivo de vídeo.
  - **RF01.1:** El sistema reescalará el vídeo a la resolución de salida deseada por el usuario.
  - **RF01.2:** El sistema generará eventos según el umbral que determine el usuario.
  - **RF01.3:** El sistema devolverá al usuario un conjunto de archivos entre los que se encuentran dos ficheros de texto, un primero con todos los eventos, y otro con los parámetros que ha seleccionado el usuario en su ejecución junto a la salida de la terminal, por último, el sistema devuelve un vídeo simulando el vídeo original a través de un sensor DVS.
- **RF02:** El sistema permitirá convertir un fichero en formato de vídeo a un archivo binario con los fotogramas del vídeo original.
- **RF03:** El sistema permitirá estimar las posiciones de la cámara a partir de un vídeo en formato binario.
- **RF04:** El sistema permitirá integrar la información obtenida con los módulos V2E y SVO en un mismo fichero binario para su uso en EMVS.
- **RF05:** El sistema estimará la profundidad de una escena grabada con un sensor DVS.
  - **RF05.1:** El sistema seleccionará los eventos y posiciones de la cámara correspondientes al intervalo estipulado por el usuario.
  - **RF05.2:** El sistema indicará a cada punto seleccionado un valor de profundidad situado en el intervalo entre un mínimo y máximo de profundidad estipulado por el usuario, y además, teniendo en cuenta un número de capas seleccionadas por el usuario.

- **RF05.3:** El sistema devolverá una imagen a color con las estimaciones de profundidad del EMVS.
- **RF06:** El sistema permitirá al usuario obtener una reconstrucción 3D de una escena grabada por un sensor DVS.
  - **RF06.1:** El sistema obtendrá los eventos que se incluirán para la nube de puntos de un intervalo estipulado por el usuario.
  - **RF06.2:** El sistema pondrá cada punto en una capa según la profundidad estimada.
  - **RF06.3:** El sistema devolverá una nube de puntos representando la reconstrucción 3D del entorno generada por el EMVS.

#### 4.2.2. No funcionales

Los requisitos no funcionales definen aquellas restricciones o propiedades que debería poseer el sistema para ser considerado bueno.

- **RNF01 Usabilidad:** Se proporcionará un resultado fácilmente comprensible por el usuario haciendo uso de un código de colores para la estimación de la profundidad.
- **RNF02 Disponibilidad:** El sistema estará disponible para los usuarios una vez se instale en el sistema.
- **RNF03 Portabilidad:** Al sistema se podrá acceder desde cualquier ordenador.
- **RNF04 Mantenibilidad:** El mantenimiento del sistema será sencillo debido a la modularidad de este y la documentación detallada.
- **RNF05 Accesibilidad:** El sistema estará disponible para todo aquel que lo requiera sin ningún tipo de credencial.
- **RNF06 Calidad:** El sistema procurará dar unos resultados de calidad con la reconstrucción 3D del entorno captado.
- **RNF07 Facilidad de uso:** Se procurará que el sistema garantice la sencillez de su uso lo suficiente para el usuario que lo requiera.
- **RNF08 Seguridad:** Se garantizará la protección de datos del usuario, asegurando que no pasen a terceros.
- **RNF09 Software:** Se garantizará que el Software se encuentre actualizado a la versión correcta requerida por el sistema de forma que se asegure un correcto funcionamiento y cumpla los requisitos de este.

- **RNF10 Hardware:** Se procurará que los elementos del Hardware, como la GPU, se encuentre actualizado para el correcto funcionamiento del sistema.

### 4.3. Diagrama de Flujo

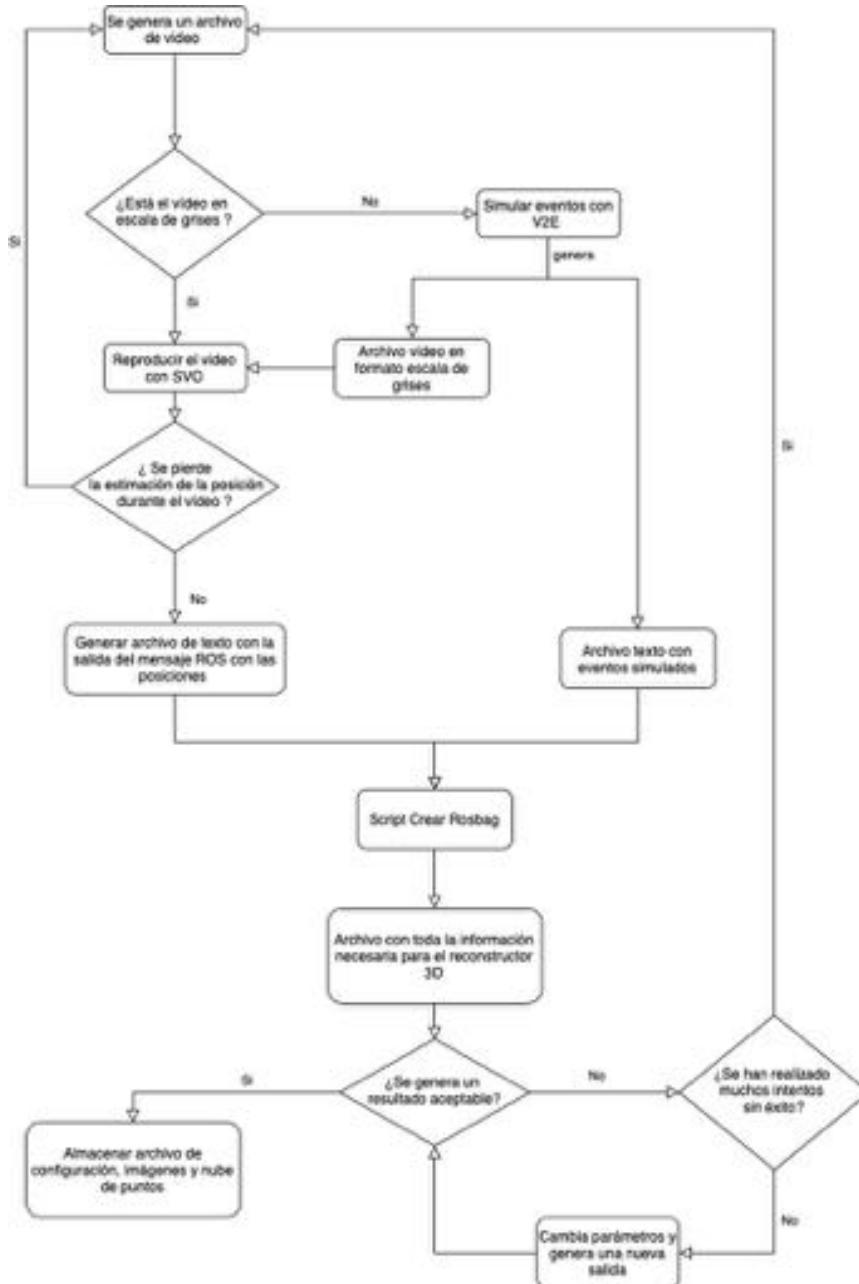


Figura 4.2: Diagrama de Flujo

Para mostrar de manera más clara el proceso a seguir para pasar de un vídeo grabado desde una cámara tradicional a la propia reconstrucción

3D por eventos con un diagrama de flujo 4.2, de forma vertical de arriba a abajo con el objetivo de llegar a conseguir un resultado aceptable en el reconstructor 3D.

Consideramos un resultado aceptable aquel que sea capaz de mostrar una imagen de profundidad con capas de colores reconocibles, donde no existan zonas de la imagen en las que haya puntos de todas las capas de profundidad, o una gran cantidad de puntos en la imagen donde no sea capaz de reconocer los objetos captados en el vídeo.

# Capítulo 5

## Implementación

En este capítulo se explicarán las modificaciones realizadas en cada uno de los sistemas detallados en el capítulo de Diseño [4] para el correcto funcionamiento del proyecto, incluyendo los distintos archivos de código que se han tenido que implementar para tratar los datos de salida de los módulos para que los siguientes pudieran tratar con la información necesaria. Además al final se detalla un manual de usuario explicando cómo se ha instalado y se usa el proyecto, liberado al completo en Github en el repositorio [41], donde se encuentra el propio manual de usuario además de los distintos sistemas y enlaces a los repositorios de los autores.

### 5.1. V2E

El primero de los sistemas de los que hacemos uso es el simulador de eventos V2E Ref1, debido a la ausencia del propio sensor de eventos para la realización del proyecto. Pero aprovechando las ventajas que otorga disponer de los fotogramas originales para el uso en métodos tradicionales, como la estimación de la cámara que veremos más adelante.

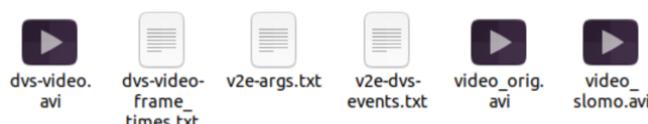


Figura 5.1: Salida de la ejecución del simulador V2E

Los ficheros de salida que se generan en una ejecución del simulador V2E [5.1] son tres archivos de vídeo, todos con la resolución pasada por argumento, el vídeo con nombre “video\_orig.avi” [5.2] es el archivo de vídeo original [5.3] pasado a escala de grises, el cual usaremos más tarde en el siguiente módulo

SVO, “video\_slomo.avi” es el vídeo original tras pasarlo por la red neuronal Super-SloMo interpolando fotogramas con el factor pasado por parámetro. Por último el vídeo “dvs-video.avi” [5.4] muestra una representación de la simulación de eventos acumulados, con fondo gris y cada evento representado en blanco o negro dependiendo de su polaridad durante un periodo de tiempo fijo (solo para visualización). Además se obtienen tres ficheros de texto, uno con los tiempos en los que se generan cada evento DVS “dvs-video-frame\_times.txt”, otro que almacena los parámetros de entrada al igual que los mensajes de salida por terminal mientras se ejecuta el simulador y finalmente, “v2e-dvs-events.txt” que almacena cada uno de los eventos generados con sus respectivas posiciones, marca de tiempo y polaridad.



Figura 5.2: Fotograma en escala de grises del video original generado por V2E



Figura 5.3: Fotograma RGB del vídeo grabado por la cámara del iPhone



Figura 5.4: Fotograma con eventos positivos (blanco) y negativos (negro) del vídeo DVS generado por V2E

## 5.2. SVO

Además de los eventos, el reconstructor 3D necesita de las posiciones de la cámara para la correcta localización de los eventos en el espacio. Para ello usamos el sistema SVO [50], que es capaz de estimar la posición de

la cámara a través de los fotogramas en formato de escala de grises. Si queremos usar nuestros propios archivos de vídeo para obtener las posiciones de la cámara deberemos cambiar los parámetros de calibración con los datos correctos de la cámara (resolución, distancia focal y punto central de la cámara) especificados en la carpeta `svo_ros/param` en un archivo con formato YAML como aparece en la Figura 5.5.

```
1 cam_model: ATAN
2 cam_width: 640
3 cam_height: 480
4 cam_fx: 2090.4
5 cam_fy: 2094.4
6 cam_cx: 320
7 cam_cy: 240
```

Figura 5.5: Archivo YAML con información sobre la cámara

Una vez tenemos correctos los parámetros de la cámara debemos reproducir el vídeo con la función `rosbag play` de ROS, este vídeo debe estar en escala de grises para que funcione con el sistema SVO además de ser un fichero `.bag` para poder reproducirlo. Como tenemos el vídeo generado en escala de grises gracias al V2E solamente hay que pasarlo a formato `bag` para poder reproducirlo con ROS, para ello hacemos uso del script `video2bag` 5.6 de python, ayudándonos de la respuesta dada a la pregunta en [51], que convierte nuestro fichero de vídeo en un fichero binario con formato `bag` listo para usarse en ROS y estimar las posiciones de la cámara con SVO.

```

1  import time, sys, os
2  from ros import rosbag
3  import roslib, rospy
4  roslib.load_manifest('sensor_msgs')
5  from sensor_msgs.msg import Image
6
7  from cv_bridge import CvBridge
8  import cv2
9
10 TOPIC = 'camera/image_raw'
11
12 def CreateVideoBag(videopath, bagname):
13     '''Creates a bag file with a video file'''
14     print("Va a crear el video")
15     bag = rosbag.Bag(bagname, 'w')
16     cap = cv2.VideoCapture(videopath)
17     cb = CvBridge()
18     prop_fps = cap.get(cv2.CAP_PROP_FPS)
19     if prop_fps != prop_fps or prop_fps <= 1e-2:
20         print ("Warning: can't get FPS. Assuming 24.")
21         prop_fps = 24
22     ret = True
23     frame_id = 0
24     while(ret):
25         ret, frame = cap.read()
26         if not ret:
27             break
28         stamp = rospy.rostime.Time.from_sec(float(frame_id) / prop_fps)
29         frame_id += 1
30         image = cb.cv2_to_imgmsg(frame, encoding="bgr8")
31         image.header.stamp = stamp
32         image.header.frame_id = "camera"
33         bag.write(TOPIC, image, stamp)
34     cap.release()
35     bag.close()
36
37 if __name__ == "__main__":
38     if len( sys.argv ) == 3:
39         CreateVideoBag(*sys.argv[1:])
40     else:
41         print( "Usage: video2bag videofilename bagfilename" )

```

Figura 5.6: Script video2bag

### 5.3. EMVS

Finalmente el último de los sistemas usados en el proyecto es el propio reconstructor 3D EMVS: Event-based Multi-View Stereo [52].

El algoritmo se nutre de la información de los eventos, los parámetros de la cámara con la que se ha grabado el vídeo y de las posiciones de la cámara conocidas durante el vídeo, todo ello encontrado en un mismo fichero binario con formato bag que ROS es capaz de leer y extraer su información.

### 5.4. Crear\_rosbag

Para crear el fichero necesario para EMVS, juntando toda la información extraída de los distintos simuladores, se ha diseñado un nuevo paquete de

ROS capaz de extraer la información de dos ficheros de texto y un yaml.

- Fichero de texto donde se encuentran todos los eventos generados por el V2E con el fichero “v2e-dvs-events.txt”.
- Fichero de texto con las posiciones publicadas por el SVO en el topic /svo/pose, que se almacenan redirigimos la salida de la terminal con la orden “rostopic echo /svo/pose svo\_posiciones.txt”.
- Fichero yaml usado para el SVO con la información de la cámara.

En el siguiente diagrama de flujo 5.7 se explica la idea general de la función del paquete Crear\_rosbag, empezando por leer y almacenar del archivo “svo\_posiciones.txt” las marcas de tiempo, las posiciones y orientaciones de la cámara, y del fichero YAML los parámetros de calibración de la misma. A continuación, se recorre cada una de las marcas de tiempo que se han leído, se escribe en el fichero binario la posición y orientación de la cámara correspondientes a esa marca de tiempo, y también la información de la cámara. Además, se recorre el fichero de eventos, escribiendo en el fichero binario solamente los eventos con la marca de tiempo que se encuentre dentro del intervalo entre la marca de tiempo de la iteración anterior y la actual. Una vez se termina de recorrer las marcas de tiempo se sale del bucle y se finaliza la escritura del archivo.

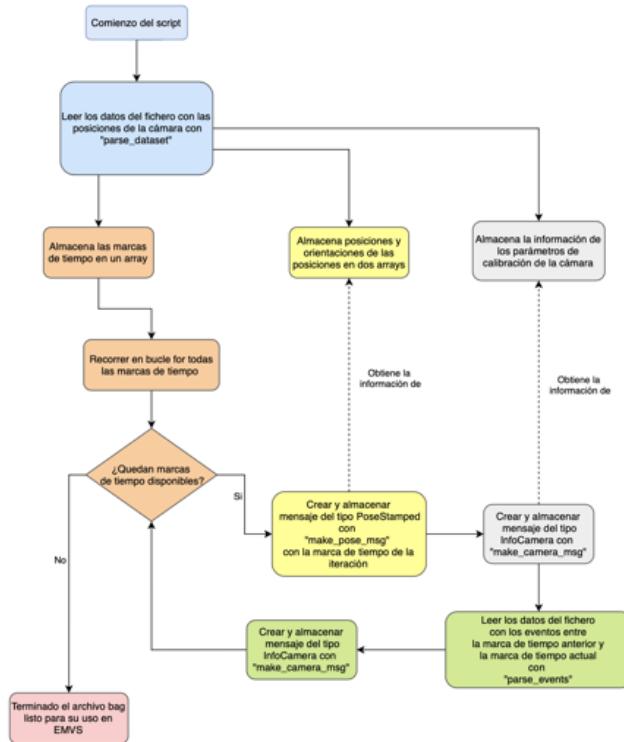


Figura 5.7: Diagrama flujo de Crear\_rosbag

Este paquete tiene un esqueleto similar al de otros simuladores como el EMVS. Contiene un fichero CMakeList.txt [5.8] en el que se especifica los distintos paquetes que se requieren para su funcionamiento (rospy, dvs\_msgs), un fichero package.xml [5.9] donde se especifican los paquetes a compilar antes del propio crear\_rosbag y un fichero .launch [5.10] para poder ejecutarlo con la orden de ROS roslaunch.

```

1 cmake_minimum_required(VERSION 2.8.3)
2 project(crear_rosbag)
3
4 find_package(catkin REQUIRED COMPONENTS
5   rospy
6   dvs_msgs
7 )
8
9 catkin_package(
10 )
11
12 ## Uncomment this if the package has a setup.py. This macro ensures
13 ## modules and global scripts declared therein get installed
14 ## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
15 #catkin_python_setup()
16
17 include_directories(
18   ${catkin_INCLUDE_DIRS}
19 )

```

Figura 5.8: CMakeList.txt del paquete crear\_rosbag

```

1  xml version="1.0"?
2  <package>
3    <name>crear_rosbag</name>
4    <version>0.0.6</version>
5    <description>Crear rosbag</description>
6
7    <maintainer email="joseadr@correo.ugr.es">Jose Antonio Diaz Ramirez</maintainer>
8    <license>GNU GPL</license>
9
10   <buildtool_depend>catkin</buildtool_depend>
11   <build_depend>rospy</build_depend>
12   <build_depend>rospkg</build_depend>
13   <build_depend>dvs_msgs</build_depend>
14   <build_depend>geometry_msgs</build_depend>
15   <build_depend>sensor_msgs</build_depend>
16   <!--<build_depend>cv_bridge</build_depend>-->
17   <run_depend>rospy</run_depend>
18   <run_depend>rospkg</run_depend>
19   <run_depend>dvs_msgs</run_depend>
20   <run_depend>geometry_msgs</run_depend>
21   <run_depend>sensor_msgs</run_depend>
22   <!--<run_depend>cv_bridge</run_depend> -->
23
24 </package>
```

Figura 5.9: package.xml del paquete crear\_rosbag

```

1 <launch>
2   <!-- DVS simulator using a sequence of images -->
3   <node name="" pkg="crear_rosbag" type="crear_rosbag.py" output="screen">
4
5   </node>
6
7 </launch>
```

Figura 5.10: Fichero launch del paquete crear\_rosbag

Con el paquete ya listo para su correcta compilación, se crea el script de python “crear\_rosbag.py”, que realiza la unión de la información de los distintos ficheros de texto en un solo binario con formato bag, pasando esta información con sus respectivos formatos adecuados para el entendimiento de ROS y el reconstructor 3D EMVS. Para comenzar con el fichero crear\_rosbag.py se deben definir las importaciones de bibliotecas necesarias para su posterior uso [\[5.11\]](#)

```

9   import rospy
10  import rospkg
11  import rosbag
12  import yaml
13
14  import os.path
15  import time
16  import numpy as np
17  from os.path import join
18
19  from dvs_msgs.msg import Event, EventArray
20  from geometry_msgs.msg import PoseStamped
21  from sensor_msgs.msg import CameraInfo
```

Figura 5.11: Librerías necesarias para funcionamiento del paquete crear\_rosbag

A continuación, se definen los distintos métodos de los que se harán en

la función `__main__`, función que se llama siempre que se ejecuta el programa, y se irán explicando conforme se llaman en el propio `__main__`. En la Figura 5.12 se pueden observar las distintas llamadas a funciones de ROS para establecer el formato de los mensajes y acceso al directorio donde se encuentran los archivos necesarios. También se realiza la llamada a la función “`parse_dataset`”, función que reúne la información necesaria de la cámara, tanto las estimaciones de posiciones, como los parámetros de calibración.

```

170 if __name__ == '__main__':
171     rospack = rospkg.RosPack()
172     package_dir = rospack.get_path('crear_rosbag')
173
174     times,positions,orientations,cam = parse_dataset(package_dir,package_dir)
175
176     ns = rospy.get_param('ns', '/dvs')
177
178     camera_info_msg = make_camera_msg(cam)
179     pose_topic = '/optitrack/davis'
180     camera_info_topic = '{}{}/camera_info'.format(ns)
181     event_topic = '{}{}/events'.format(ns)
182
183     init_time = times[0]
184
185     # Debe estar en el mismo directorio
186     bag = rosbag.Bag(join(package_dir, '{}-{}.bag'.format("prueba_1", time.strftime("%Y%m%d-%H%M%S"))), 'w')
187
188     bag.write(topic=pose_topic, msg=make_pose_msg(positions[0], orientations[0], init_time), t=init_time)
189
190     last_pub_event_timestamp = init_time
191     events = []
192     limite = 7
193     delta_event = rospy.Duration(1.0 / 300)
194

```

Figura 5.12: Comienzo del `__main__` de `crear_rosbag.py`

La función “`parse_dataset`” se implementa tomando como referencia<sup>1</sup> empieza en Figura 5.13, inicializando las variables donde se almacenarán los datos de marcas de tiempo, posiciones y orientaciones, y cargando los datos del archivo YAML con los parámetros de calibración de la cámara, almacenando la resolución (alto y ancho), distancias focales (fx,fy) y el punto central (cx,cy) como se observa en la Figura 5.14.

```

68 def parse_dataset(cam_dir,dataset_dir):
69
70     # Parse camera calibration
71     cam_file = open('{}/{}/camera_prueba.yaml'.format(cam_dir,dataset_dir))
72     cam_data = yaml.safe_load(cam_file)
73
74     image_data = {}
75     tiempos=[]
76     posiciones=[]
77     orientaciones=[]
78
79     width = cam_data['cam_width']
80     height = cam_data['cam_height']
81     fx = cam_data['cam_fx']
82     fy = cam_data['cam_fy']
83     cx = cam_data['cam_cx']
84     cy = cam_data['cam_cy']
85
86     cam = [width, height, fx, fy, cx, cy]

```

Figura 5.13: Comienzo función `parse_dataset` de `crear_rosbag.py`

Figura 5.14: Asignación de datos de la cámara para su salida en la función `parse_dataset`

<sup>1</sup>[https://github.com/uzh-rpg/rpg\\_davis\\_simulator/blob/50e84063fffb9de609091df2f77b2ad52c3008fa/src/dvs\\_simulator\\_py/dvs\\_simulator.py](https://github.com/uzh-rpg/rpg_davis_simulator/blob/50e84063fffb9de609091df2f77b2ad52c3008fa/src/dvs_simulator_py/dvs_simulator.py)

A continuación, se entra en el fichero de texto con la salida del topic /svo/pose [5.16](#) y se van almacenando los datos como se observa en [5.15](#), teniendo en cuenta la estructura del topic de tipo PoseStamped.

```
78     with open('%s/svo_posicion.txt' % dataset_dir) as fichero:
79         linea = fichero.readline()
80         contador = 1
81         while linea:
82             if contador==4:
83                 secs = int(linea.split(':')[1][1])
84                 #print(secs)
85             elif contador == 5:
86                 nsecs = int(linea.split(':')[1][1])
87                 #print(nsecs, "\n")
88                 tiempo = rospy.Time(secs,nsecs)
89                 tiempos.append(tiempo)
90             elif contador == 10:
91                 posicion=[]
92                 posicion.append(float(linea.split(':')[1][1]))
93             elif contador == 11:
94                 posicion.append(float(linea.split(':')[1][1]))
95             elif contador == 12:
96                 posicion.append(float(linea.split(':')[1][1]))
97                 posiciones.append(posicion)
98             elif contador == 14:
99                 orientacion=[]
100                orientacion.append(float(linea.split(':')[1][1]))
101            elif contador == 15:
102                orientacion.append(float(linea.split(':')[1][1]))
103            elif contador == 16:
104                orientacion.append(float(linea.split(':')[1][1]))
105            elif contador == 17:
106                orientacion.append(float(linea.split(':')[1][1]))
107                orientaciones.append(orientacion)
108            elif contador == 19:
109                contador = 0
110                linea = fichero.readline()
111                contador += 1
```

Figura 5.15: Tratamiento del archivo con las posiciones guardadas de la función parse\_dataset

```

1 header:
2   seq: 0
3   stamp:
4     secs: 1
5     nsecs: 600000000
6   frame_id: "/cam"
7 pose:
8   pose:
9     position:
10    x: -0.0005051626696041008
11    y: -0.0003814055840161897
12    z: 0.006104002145171258
13   orientation:
14    x: 0.9999601618321416
15    y: 0.00877657282699884
16    z: 0.0005486681502216052
17    w: 0.0015314964284824661
18 covariance: [4.65701449823234e-310, 6.9451541083709e-310, 4.65701449786713e-310,
4.65701449786554e-310, 4.65701449823234e-310, 6.94515410838357e-310, 4.65701449786713e-310,
4.65701449786554e-310, 4.65701449823234e-310, 6.9451541083456e-310, 4.65701449786713e-310,
4.65701449786554e-310, 4.65701449823234e-310, 6.94515400975384e-310, 4.65701449786713e-310,
4.65701449786554e-310, 4.65701449823234e-310, 4.65701283103667e-310, 4.65701449786713e-310,
4.65701449786554e-310, 4.65701449823234e-310, 6.94515432746097e-310, 4.65701449786713e-310,
4.65701449786554e-310, 4.65701449823234e-310, 6.9451541083709e-310, 4.65701449786713e-310,
4.65701449786554e-310, 4.65701449823234e-310, 6.94515410838357e-310, 4.65701449786713e-310,
4.65701449786554e-310, 4.65701449823234e-310, 6.9451540097918e-310, 4.65701449786713e-310,
4.65701449786554e-310]
19 ---

```

Figura 5.16: Ejemplo de salida de una posición de la cámara publicada por el topic /svo/pose

Para ello vamos leyendo cada línea para almacenar la información necesaria, siguiendo la Figura 5.16, las marcas de tiempo vienen dadas en las líneas 4 y 5, las coordenadas 3D de la posición situadas en la línea 10, 11 y 12 y por último la orientación en las líneas 14, 15 y 16. Al tener siempre la misma estructura, se recorren todos los mensajes del archivo, almacenando en varios arrays todas las marcas de tiempo, posiciones y orientaciones para su posterior escritura en el fichero binario bag.

```

25   def make_camera_msg(cam):
26     camera_info_msg = CameraInfo()
27     width, height = cam[0], cam[1]
28     fx, fy = cam[2], cam[3]
29     cx, cy = cam[4], cam[5]
30     camera_info_msg.distortion_model="plumb_bob"
31     camera_info_msg.width = width
32     camera_info_msg.height = height
33     camera_info_msg.K = [fx, 0, cx,
34                           0, fy, cy,
35                           0, 0, 1]
36
37     camera_info_msg.D = [0, 0, 0, 0]
38
39     camera_info_msg.P = [fx, 0, cx, 0,
40                           0, fy, cy, 0,
41                           0, 0, 1, 0]
42     return camera_info_msg

```

Figura 5.17: Función make\_camera\_msg

Continuando después de la llamada a la función parse\_dataset, se asigna la primera de las marcas de tiempo como el tiempo inicial. Se almacena en una variable el mensaje pasado al formato del topic CameraInfo con la función “make\_camera\_msg” 5.17. Se crea el fichero bag destinado a usarse en el EMVS y se le añade la primera de las posiciones con la marca de tiempo inicial. El reconstructor 3D, lee del topic “optitrack/davis” para las posiciones de la cámara, esto es porque los conjuntos de datos que tienen los autores de EMVS grabaron los datos con un sistema OptiTrack de Natural-Point [53], sistemas que se usan para la captura de movimiento con altísima precisión. Entonces, para facilitar su uso, se asigna el mismo nombre al topic que generamos con las estimaciones de la cámara dadas por SVO.

```
46     def make_pose_msg(position, orientation, timestamp):
47         pose_msg = PoseStamped()
48         pose_msg.header.stamp = timestamp
49         pose_msg.header.frame_id = '/cam'
50         pose_msg.pose.position.x = position[0]
51         pose_msg.pose.position.y = position[1]
52         pose_msg.pose.position.z = position[2]
53         pose_msg.pose.orientation.x = orientation[0]
54         pose_msg.pose.orientation.y = orientation[1]
55         pose_msg.pose.orientation.z = orientation[2]
56         pose_msg.pose.orientation.w = orientation[3]
57     return pose_msg
```

Figura 5.18: Función make\_pose\_msg

Para crear el mensaje de la posición se llama a “make\_pose\_msg” 5.18, adquiere el formato del mensaje PoseStamped, y se asignan marca de tiempo, posición y orientacion. Se recorren todas las marcas de tiempo 5.19, para cada una de ellas se escribirá la información de la posición y de la cámara con las funciones “make\_pose\_msg” y “make\_camera\_msg” como se ha visto anteriormente.

```

196     for frame_id in range(1, len(times)):
197         print("Va por:", frame_id, "(", times[frame_id],") de ",len(times))
198         timestamp = times[frame_id]
199
200         bag.write(topic=pose_topic, msg=make_pose_msg(positions[frame_id], orientations[frame_id], timestamp), t=timestamp)
201
202         bag.write(topic=camera_info_topic, msg=camera_info_msg, t=timestamp)
203
204         #print("Llego aqui")
205
206         events, limite = parse_events(package_dir,timestamp,last_pub_event_timestamp,limite)
207         #print("Llego aqui2")
208         # publish events
209
210         if timestamp - last_pub_event_timestamp > delta_event:
211             events = sorted(events, key=lambda e: e.ts)
212             event_array = EventArray()
213             event_array.header.stamp = timestamp
214             event_array.width = cam[0]
215             event_array.height = cam[1]
216             event_array.events = events
217
218             bag.write(topic=event_topic, msg=event_array, t=timestamp)
219
220             events = []
221             last_pub_event_timestamp = timestamp
222
223     bag.close()
224     rospy.loginfo("Finished writing rosbag")

```

Figura 5.19: Bucle recorriendo cada una de las marcas de tiempo

Pero además se accede al archivo donde se almacenan los eventos con la función “parse\_events” [5.20], que recorre los eventos, almacenando solamente los que tienen una marca de tiempo anterior a la actual y posterior a la marca de tiempo de la anterior iteración. En caso de ser la primera iteración en el bucle se empieza en la línea 7 a leer ya que las 6 primeras líneas del fichero “v2e-dvs-events.txt” [5.21] no corresponden a eventos.

```

137     def parse_events(dataset_dir,timestamp,last_pub,limite):
138         eventos = []
139         with open('%s/v2e-dvs-events.txt' % dataset_dir) as fichero:
140             linea = fichero.readline()
141             contador = 1
142             print("Empieza a leer eventos")
143             while linea:
144                 if contador<limite:
145                     contador += 1
146                     if(contador==limite-1):
147                         print("Se ha saltado la cabecera = ", limite)
148                 else:
149                     time = float(linea.split(' ')[0])
150                     if( rospy.Time.from_sec(time)<timestamp and rospy.Time.from_sec(time) > last_pub ):
151                         x = int(linea.split(' ')[1])
152                         y = int(linea.split(' ')[2])
153                         polarity = bool(linea.split(' ')[2])
154                         eventos.append(make_event(x, y, time, polarity))
155                         contador += 1
156                     elif(rospy.Time.from_sec(time) > timestamp):
157                         print("Tamaño de eventos es",len(eventos))
158                         limite = contador
159                         break
160                     else:
161                         contador += 1
162
163             linea = fichero.readline()
164
165     return eventos, limite

```

Figura 5.20: Función parse\_events

```

#!events.txt
# This is a text DVS created by v2e (see https://github.com/SensorsINI/v2e)
# Format is time (float s), x, y, polarity (0=off, 1=on) as specified at http://rpg.ifii.uzh.ch/
# Creation time: 01:09AM May 12 2021
# Creation time: System.currentTimeMillis() 1620774550480
# User name: parallels
0.0013900846242904663 303 123 1
0.0013900846242904663 147 259 1

```

Figura 5.21: Comienzo archivo v2e-dvs-events.txt

Con la intención de recorrer de forma más eficiente el archivo, guardamos siempre el número de la línea del último evento seleccionado, para así empezar con las comprobaciones de tiempos desde esa línea y no siempre desde el comienzo del archivo, evitando así leer todos los eventos en cada intervalo de tiempo.

```

129  def make_event(x, y, ts, pol):
130      e = Event()
131      e.x = x
132      e.y = y
133      e.ts = rospy.Time(secs=ts)
134      e.polarity = pol
135      return e

```

Figura 5.22: Función make\_events

En cada uno de los eventos se llama a la función “make\_events” 5.22 se adquiere el formato Event, y se le asignan los valores del píxel donde se genera el evento (x,y) la marca de tiempo en la que se generó y la polaridad (positivo o negativo). Una vez se termina la escritura del fichero bag, se han asignado todos los eventos transcurridos entre dos posiciones de cámaras a la segunda de ellas, basándonos en la idea del repositorio Ref7 que simula el comportamiento del sensor DVS.

```

--min_depth=0.7
--max_depth=0.75
--start_time_s=25.0
--stop_time_s=25.1
--dimZ=5

```

Figura 5.23: Ejemplo parámetros configuración EMVS

Ya con el fichero bag, se puede ejecutar con el reconstructor 3D usando un fichero de configuración como por ejemplo el de la 5.23 en el que se pueden modificar distintas variables como el tiempo del fichero en el que se quiere realizar la reconstrucción 3D, filtros para reducir el ruido o el número de capas de profundidad en el que se calcula la estimación, todo ello explicado

más a fondo en el capítulo posterior de experimentos. Por último, para poder visualizar la nube de puntos que genera el reconstructor 3D EMVS se ha instalado la herramienta pcl.viewer a través de la terminal usando el comando: “sudo apt-get install pcl-tools” y se puede ver el fichero con el comando por terminal “pcl\_viewer pointcloud.pcd” siendo pointcloud.pcd la nube de puntos generada.

## 5.5. Manual de usuario

Para comenzar a instalar la totalidad del proyecto para su uso en Ubuntu 20.04 se requiere la instalación del framework ROS Noetic y el uso de Anaconda siguiendo los pasos que se indican en <sup>2</sup> y <sup>3</sup>, y a continuación se especificarán las instrucciones de instalación de cada sistema uno a uno. Se pueden encontrar en el repositorio de mi trabajo [\[41\]](#) también.

### 5.5.1. V2E

#### Instalación

Comenzando por el V2E seguiremos los pasos de instalación de los autores encontrados en el propio repositorio Ref1, creándole un entorno de anaconda para su correcto funcionamiento sin conflictos por incompatibilidades e instalando las dependencias necesarias.

#### Uso

Se activa el entorno virtual de anaconda creado para el simulador V2E y se ejecuta con los parámetros deseados, explicados en el repositorio [\[54\]](#).

### 5.5.2. Video2Bag

#### Instalación

Una vez descargado el script video2bag se requiere instalar por terminal los siguientes paquetes:

- pip install rospkg
- pip install pycryptodomex

---

<sup>2</sup><http://wiki.ros.org/noetic/Installation/Ubuntu>

<sup>3</sup><https://www.digitalocean.com/community/tutorials/how-to-install-the-anaconda-python-distribution-on-ubuntu-20-04-es>

- pip install gnupg
- pip install opencv-python

Una vez completado, se siguen los pasos especificados <sup>4</sup>, donde se crea un espacio de trabajo catkin con las librerías necesarias que facilitan el uso de OpenCV en ROS <sup>5</sup>.

## Uso

Situamos el entorno de la terminal en ROS accediendo al archivo /ros/noetic/setup.bash y a continuación en el entorno de trabajo Catkin que hemos creado anteriormente, accediendo al setup.bash generado en la carpeta devel y ejecutamos el script de forma habitual con “python3 video2bag videofilename bagfilename” donde el archivo “videofilename.es” el video que queremos convertir a binario, que se almacenará en un archivo con formato bag cuyo nombre especifiquemos en “bagfilename”.

### 5.5.3. SVO

#### Instalación

Para la instalación del SVO en ROS se siguen las instrucciones paso a paso detalladas en la wiki del propio repositorio, dejando dos espacios de trabajo Catkin separados, uno para las herramientas Sophus <sup>6</sup>, Fast <sup>7</sup> y opcionalmente, g2o y el otro espacio de trabajo con vikit y el propio SVO.

## Uso

Tras configurar el módulo SVO su ejecución sigue los mismos pasos que un paquete de ROS, es necesario ejecutar “roscore” en una terminal distinta para inicializar ROS, a continuación nos situamos en el entorno del espacio de trabajo donde se encuentre SVO accediendo al setup.bash en el directorio devel, y ejecutamos en la misma terminal el comando “roslaunch svo\_ros test\_rig3.launch” como se especifica en la wiki del repositorio. Para ejecutar vídeos con distinta resolución antes de ejecutar se debe cambiar en el archivo YAML de los parámetros de la cámara en el directorio svo\_ros/param y volver a compilar el entorno catkin con el comando “catkin build svo\_ros”. Para guardar la información necesaria para el EMVS, en una nueva terminal

---

<sup>4</sup><https://www.programmersought.com/article/85247550924/>

<sup>5</sup>[http://wiki.ros.org/vision\\_opencv](http://wiki.ros.org/vision_opencv)

<sup>6</sup><https://github.com/strasdat/Sophus.git>

<sup>7</sup><https://github.com/uzh-rpg/fast.git>

ejecutamos el comando “rostopic echo /svo/pose” redirigiendo la salida a un archivo de texto antes de reproducir el fichero binario.

#### 5.5.4. Crear\_rosbag

##### Instalación

Primero han de instalarse los drivers<sup>8</sup> de los sensores DVS/DAVIS para poder utilizar las distintas herramientas y estructuras de datos propias de estos sensores, y así después clonando mi repositorio personal<sup>9</sup> dentro del mismo espacio de trabajo catkin donde se configuraron los drivers anteriores.

##### Uso

```
70      # Parse camera calibration
71      cam_file = open('%s/camera_prueba.yaml' % cam_dir)
```

Figura 5.24: Nombre fichero calibración de la cámara

Cada vez que queramos usar el módulo “crear\_rosbag” con ficheros distintos, se deben cambiar las siguientes líneas si se desea usar otro fichero distinto. La línea 71 [5.24] donde se especifica el nombre del fichero YAML con los parámetros de calibración de la cámara, línea 78 [5.25] para indicar el nombre del archivo de texto donde se almacena la salida del SVO, línea 139 [5.26]

```
78      with open('%s/svo_posicion.txt' % dataset_dir) as fichero:
```

Figura 5.25: Nombre fichero posiciones sacadas por SVO

```
139      with open('%s/v2e-dvs-events.txt' % dataset_dir) as fichero:
```

Figura 5.26: Nombre fichero con el conjunto de eventos

para indicar el nombre del archivo con el conjunto de todos los eventos (normalmente al hacer uso de V2E siempre será el mismo) y por último la línea 187 [5.27] para indicar el nombre de salida del fichero bag generado. Una vez hecho los cambios se vuelve a compilar el paquete crear\_rosbag con catkin y se ejecuta el comando “roslaunch crear\_rosbag prueba.launch”.

<sup>8</sup>[https://github.com/uzh-rpg/rpg\\_dvs\\_ros](https://github.com/uzh-rpg/rpg_dvs_ros)

<sup>9</sup>[https://github.com/Joseadr99/TFG-3D\\_reconstruction/tree/main/crear\\_rosbag](https://github.com/Joseadr99/TFG-3D_reconstruction/tree/main/crear_rosbag)

```
187     bag = rosbag.Bag(join(package_dir, '{}-{}.bag'.format("prueba_1", time.strftime("%Y%m%d-%H%M%S"))), 'w')
```

Figura 5.27: Nombre del fichero binario de salida con formato bag

### 5.5.5. EMVS

#### Instalación

Se siguen los pasos de instalación cambiando el nombre de la distribución kinetic por noetic en los comandos y se sigue el cambio realizado en CMakeList.txt explicado en la anterior sección [6.4.4] detallado en la [6.13]. Además son necesarios la instalación de distintos paquetes para su correcta compilación, vision\_opencv [10], geometry [11], que para su correcto funcionamiento se han requerido también eigen\_catkin [12] y eigen\_checks [13], el paquete pcl\_catkin [14], un wrapper que permite la compatibilidad de la librería Numpy de python en C++ llamado “cnpv\_catkin” [15], otros dos para los flags y mensajes de Google [16][17] y por último “minkindr” [18] que proporciona las herramientas necesarias para operar con transformaciones simples en quaterniones.

#### Uso

Se ejecuta con el comando de ROS rosrun tal que “rosrun mapper\_emvs run\_emvs”, además se le añaden los dos parámetros necesarios –bag\_filename = ruta del fichero con formato bag y –flagfile = ruta con el archivo de configuración donde se especifican los distintos parámetros como el tiempo o las capas de profundidad.

---

<sup>10</sup>[http://wiki.ros.org/vision\\_opencv](http://wiki.ros.org/vision_opencv)

<sup>11</sup><https://github.com/ros/geometry>

<sup>12</sup>[https://github.com/ethz-asl/eigen\\_catkin](https://github.com/ethz-asl/eigen_catkin)

<sup>13</sup>[https://github.com/ethz-asl/eigen\\_checks](https://github.com/ethz-asl/eigen_checks)

<sup>14</sup>[https://github.com/ethz-asl/pcl\\_catkin](https://github.com/ethz-asl/pcl_catkin)

<sup>15</sup>[https://github.com/uzh-rpg/cnpv\\_catkin](https://github.com/uzh-rpg/cnpv_catkin)

<sup>16</sup>[https://github.com/ethz-asl/gflags\\_catkin](https://github.com/ethz-asl/gflags_catkin)

<sup>17</sup>[https://github.com/ethz-asl/glog\\_catkin](https://github.com/ethz-asl/glog_catkin)

<sup>18</sup><https://github.com/ethz-asl/minkindr>

# **Capítulo 6**

## **Problemas durante el desarrollo**

### **6.1. Restricciones Hardware**

- Necesitamos una tarjeta gráfica NVIDIA para hacer uso de CUDA y poder realizar los cálculos en el simulador de eventos V2E.
- Cámara convencional capaz de grabar videos.
- La interacción humano-máquina se ha desarrollado a través de la terminal de Linux, por lo que se requiere de un teclado.

### **6.2. Restricciones Software**

- Es necesario tener ROS Noetic instalado para ejecutar el proyecto.
- ROS Noetic está disponible en el sistema operativo Ubuntu 20.04, por lo que debemos tenerlo instalado.
- Anaconda con python3 para el simulador v2e.

### **6.3. Restricciones sanitarias por el Covid-19**

Con la situación sanitaria que seguimos sufriendo durante el año 2021 debido al COVID-19 han existido limitaciones que no estarían presentes de no ser por la extraordinaria situación que nos está tocando vivir:

- Inviabilidad de la recogida y uso del sensor DVS hasta unos meses avanzados en el proyecto, lo que llevó a reestructurar el trabajo original

con la búsqueda de métodos que simularan el comportamiento del sensor en vídeos tradicionales, haciendo uso de otro tipo de cámaras como la propia del móvil.

- Necesidad de tener reuniones a modo online con el tutor por las restricciones hasta muy avanzado el proyecto, donde ya si pude asistir de modo presencial a su despacho para discutir las dudas de forma más clara y sencilla.

## 6.4. Problemas técnicos

En esta sección se describirán los problemas que más han perjudicado la planificación original, dando lugar al retraso en distintas fases, como se comentarán en los siguientes puntos.

### 6.4.1. Compatibilidad de Anaconda con ROS Noetic/Melodic

Al comienzo de la búsqueda de métodos para la simulación de los eventos se intentó instalar el simulador ESIM [55] en el entorno de ROS, pero al instalarlo no paraba de dar problemas por dependencias entre paquetes y su funcionamiento era complejo, usando como entrada un fichero binario con la información del vídeo, por lo que se optó por el uso de otro de los simuladores que se investigaron, V2E, que recibía como entrada el fichero de vídeo y una instalación sencilla sin errores de compatibilidades, en detrimento de la integración en ROS, ya que se utiliza en un entorno de Anaconda.

Esto ha provocado una gran cantidad de incompatibilidades entre paquetes de ambos entornos, Anaconda y ROS, debido al uso de Python3 por parte de anaconda y por el contrario Python 2.7 en ROS, lo cual ha llevado a la reinstalación varias veces de paquetes, hasta que se consiguió solucionar aislando en un entorno virtual de Anaconda todas las dependencias necesarias para el V2E sin colisionar con ROS.

### 6.4.2. Limitaciones generadas por V2E

Como se comentaba en la sección 3.2 del Entorno operacional, el simulador V2E que uso para el trabajo puede aprovechar el funcionamiento de la GPU con CUDA, pero mi ordenador personal no dispone de una GPU NVIDIA compatible.

Esto se traducía en tiempos altos de ejecución para generar los eventos haciendo uso de la CPU, dejando durante el proceso mi ordenador inoperable, no pudiendo avanzar en ninguno de los otros apartados de los que consta este trabajo fin de grado.

Y además, solo era capaz de generar eventos convirtiendo los vídeos en resoluciones más pequeñas, en caso de mantener una resolución original de alta definición como 720p, la CPU es incapaz de simular los eventos dando error en la ejecución debido a la falta de memoria disponible.

#### 6.4.3. Servidor compartido

Como se especifica en el primer problema de Hardware [6.1], es necesario una GPU NVIDIA para procesar cualquier tipo de vídeo, por ello se hizo uso de un servidor prestado por la UGR con las especificaciones dadas en [3.3].

Al no ser el único usuario que utilizaba el servidor, compartiéndolo con otros alumnos, había días que no estaba disponible su uso debido a otras ejecuciones que se estaban llevando en ese momento, ya que cuando uno de nosotros utilizaba el servidor solía ocupar toda la memoria de la GPU incapacitando al resto.

En mi caso esto provocó retrasos de hasta una semana en poder obtener resultados del V2E, lo que implicaba no poder comprobar la viabilidad de los vídeos en SVO, que en caso de no ser válidos, se deben descartar esos vídeos y probar con otros, repitiendo el proceso de V2E que es de larga duración.

#### 6.4.4. Modificaciones de código para correcto funcionamiento en V2E, SVO y EMVS

##### V2E

Al instalarlo siguiendo los pasos del repositorio de V2E, solo permite generar vídeos con una resolución de salida determinada, en concreto 346x260 píxeles o 240x180 píxeles, esto ocurre debido al archivo de salida que se genera con formato *aedat2*<sup>1</sup> (Address Event DATA), un fichero de datos personalizado para el almacenamiento de la información con eventos.

En nuestro caso no hacemos uso del archivo *aedat2* para la reconstrucción 3D, por lo tanto optamos por comentar en el código las líneas donde se llama a la función que genera el archivo y así poder usar cualquier resolución

---

<sup>1</sup>[https://inivation.github.io/inivation-docs/Software%20user%20guides/AEDAT\\_file\\_formats.html](https://inivation.github.io/inivation-docs/Software%20user%20guides/AEDAT_file_formats.html)

para el vídeo de salida. Para ello hace falta comentar las líneas siguientes del archivo emulator.py localizado en el directorio v2ecore:

```
167     if dvs_aedat2:
168         path = os.path.join(self.output_folder, dvs_aedat2)
169         path = checkAddSuffix(path, '.aedat')
170         logger.info('opening AEDAT-2.0 output file ' + path)
171         self.dvs_aedat2 = AEDat2Output(
172             path, output_width=self.output_width,
173             output_height=self.output_height)
```

Figura 6.1: Líneas a comentar en el constructor del archivo emulator.py

```
216     if self.dvs_aedat2 is not None:
217         self.dvs_aedat2.close()
```

Figura 6.2: Líneas a comentar en la función cleanup del archivo emulator.py

```
623         if self.dvs_aedat2 is not None:
624             self.dvs_aedat2.appendEvents(events)
```

Figura 6.3: Líneas a comentar en la función generate\_events del archivo emulator.py

Una vez hechos estos cambios, podemos introducir el archivo de vídeo que queramos y reescalarlo si deseamos a cualquier resolución, o dejar la misma resolución para la simulación de los eventos y los vídeos de salida generados.

## SVO

Al instalarlo con ROS Noetic, se usa con una versión de OpenCV más moderna a la usada por los autores del repositorio, lo cual requiere de unos cambios en el código para ajustar el nuevo modo de llamada a las funciones en OpenCV, cambiando todos los “CV\_” por “cv::” .

Estos cambios se encuentran en svo/src/sparse\_img\_align.cpp en la línea 267 [6.4](#) cambiándolo por cv::Window\_autosize, svo/test/test\_feature\_detection.cpp linea 57 donde se cambia CV\_GRAY2RGB a cv::COLOR\_Gray2RGB [6.5](#).

```
267     cv::namedWindow("residuals", CV_WINDOW_AUTOSIZE);
```

Figura 6.4: Línea 267 a cambiar del archivo sparse\_img\_align.cpp

```
57     cv::cvtColor(img, img_rgb, CV_GRAY2RGB);
```

Figura 6.5: Línea 57 del archivo test\_feature\_detection.cpp

en svo\_ros/src/visualizer.cpp líneas 141 y 149 cv::FILLED en vez de CV\_FILLED [6.6] [6.7]

```
141             cv::Scalar(0,255,0), CV_FILLED);
```

Figura 6.6: Línea 141 a cambiar del archivo visualizer.cpp

```
149             cv::Scalar(0,255,0), CV_FILLED);
```

Figura 6.7: Línea 149 a cambiar del archivo visualizer.cpp

Además en los pasos de instalación con ROS se nos pide instalar un módulo llamado *vikit* [56] que contiene varias herramientas útiles para el funcionamiento del SVO, en esos archivos también hay que realizar algunos cambios para la compatibilidad de OpenCV.

Todos los archivos que se modifican a continuación se encuentran en la carpeta vikit\_common/src del módulo *vikit*, el primero de ellos se encuentra en la línea 112 de pinhole\_camera.cpp [6.8], donde cambiamos “CV\_” por “cv::”.

```
112     cv::remap(raw, rectified, undist_map1_, undist_map2_, CV_INTER_LINEAR);
```

Figura 6.8: Línea 112 del archivo pinhole\_camera.cpp

```
48     cv::Mat cvH = cv::findHomography(src_pts, dst_pts, CV_RANSAC, 2./error_multiplier2);
```

Figura 6.9: Línea 48 del archivo homography.cpp

```
237     cv::namedWindow("residuals", CV_WINDOW_AUTOSIZE);
```

Figura 6.10: Línea 237 del archivo img\_align.cpp

```
437     cv::namedWindow("residuals", CV_WINDOW_AUTOSIZE);
```

Figura 6.11: Línea 437 del archivo img\_align.cpp

A continuación en el archivo homography.cpp [6.9] hacemos el mismo cambio que en el anterior archivo en la línea 48 del código.

Y finalmente en el archivo img\_align.cpp en las líneas 237 [6.10] y 437 [6.11], donde se realiza el mismo cambio que aplicábamos en la [6.4].

Al realizar los cambios, el algoritmo SVO funciona correctamente con el ejemplo de prueba provisto en la guía de ejecución encontrada en el repositorio de SVO.

## EMVS

Al usar la distribución ROS Noetic, versión diferente a la usada por los autores del EMVS (ROS Kinetic), las únicas modificaciones necesarias para hacer funcionar EMVS en nuestra máquina virtual son, comentar el método que escribe en disco el DSI con formato npy en la línea 98 del main.cpp [6.12], ya que no podía escribir este archivo en disco y no se necesita para lo que pretendemos con el trabajo.

```
97     // Write the DSI (3D voxel grid) to disk
98     mapper.dsi_.writeGridNpy("dsi.npy");
```

Figura 6.12: Línea a comentar en el fichero main.cpp del EMVS

```
8 set(CMAKE_CXX_FLAGS "-O3 -fopenmp -std=c++14 ${CMAKE_CXX_FLAGS}")
```

Figura 6.13: Línea 8 del fichero CMakeList.txt del mapper\_emvs

Y tal como comenta el usuario Youras en el Issue<sup>2</sup> del propio repositorio EMVS, la librería de la que se hace uso para operar con el pointcloud, pcl debe ser construida con la versión de c++14 y no la 11, como originalmente está especificada en el archivo CMakeList.txt que queda de la manera que se observa en la [6.13]

---

<sup>2</sup>[https://github.com/uzhrpg/rpg\\_emvs/issues/16](https://github.com/uzhrpg/rpg_emvs/issues/16)

#### 6.4.5. Imprevistos en la experimentación (SVO y EMVS)

El gran retraso producido en la parte final del proyecto vino por culpa de la viabilidad de los vídeos para el SVO y para el reconstructor 3D EMVS.

Con respecto al SVO, los vídeos requerían de puntos característicos en los fotogramas. Para ser capaz de detectarlos durante el vídeo para estimar la posición, estos puntos suelen aparecer en los píxeles de la imagen donde existe mayor contraste con sus píxeles vecinos. Para poder estimar la posición correctamente, el número de puntos característicos debe ser grande para poder realizar una correspondencia que sea significativa entre fotogramas. Si no los hay, la estimación de la posición de la cámara no se puede realizar.

La cantidad de vídeos que no han conseguido estimar la posición sin perderse en mitad o ni siquiera empezar el proceso ha sido muy grande, siendo un imprevisto a la hora de la experimentación, teniendo que grabar unos vídeos con movimientos suaves, y a pesar de ello, la estimación no se asemeja al movimiento real seguido en muchos de los casos.

Además, una vez finalizado el módulo para integrar los distintos componentes y probar el reconstructor 3D, los resultados no eran los esperados, llevando a realizar muchas más pruebas distintas para comprobar dónde se encontraba el problema e intentar minimizarlo para conseguir mejores resultados, como se verá en el capítulo de Experimentos [7](#)

# Capítulo 7

# Experimentos

Nuestro proyecto consiste en la integración de un sistema que trate la información de los eventos de un sensor DVS, con el objetivo final de obtener una reconstrucción 3D de una escena. Para ello, nos apoyamos en el uso de distintos sistemas, uno para la simulación de eventos a partir de un vídeo con V2E, otro para la estimación de la posición de la cámara durante el vídeo con SVO, y finalmente el sistema EMVS, el reconstructo 3D a partir de la información obtenida por los otros dos sistemas anteriores.

Los conjuntos de experimentos a realizar para cada uno de los componentes serán los siguientes:

- Conjunto de vídeos del dataset de la Universidad de Zurich
- Pruebas con un tablero de ajedrez
- Pruebas con texturas extravagantes
- Pruebas con Arucos y graffitis

## 7.1. SVO

Para medir la viabilidad del sistema SVO, que estima la posición de la cámara, se han realizado una gran cantidad de vídeos con distintas texturas en el entorno, para una mejor captación de características robustas y así, obtener con mayor seguridad una estimación de la posición de la cámara. La falta de texturas en el entorno puede provocar una pérdida en la estimación de la posición proporcionada por SVO, resultando en intervalos con una estimación errónea o reiniciada a la posición inicial hasta la correcta relocalización.

Por ello, para intentar mejorar la consistencia del algoritmo SVO durante los vídeos, se fue mejorando en tres campos:

- Texturas en los vídeos
- Movimientos suaves
- Cambio de parámetros

El seguimiento de la cámara se representa como se observa en las Figuras 7.1, 7.2 y 7.3, donde se puede ver cómo el componente SVO estima las posiciones de la cámara a lo largo de un vídeo, marcando con un punto las posiciones por las que ha ido transcurriendo la cámara en el espacio, y un prisma triangular mostrando la orientación de la misma.



Figura 7.1: Comienzo de la estimación de un vídeo con SVO



Figura 7.2: Seguimiento de la estimación de un vídeo con SVO

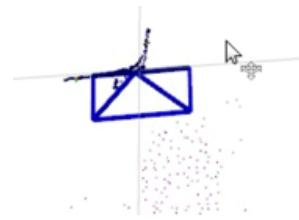


Figura 7.3: Trayectoria resultante de la estimación de un vídeo con SVO

#### 7.1.1. Prueba con vídeo del Dataset Zurich (oficina)

En este vídeo se graba una oficina con una persona sentada, donde se extraen las características en los bordes de los objetos y en la persona al iniciar el vídeo 7.4, pero encuentra dificultades una vez el movimiento de la cámara se hace más rápido, a pesar de encontrarse con prácticamente la misma escena 7.5. Al perder la estimación muy cercano al comienzo del vídeo, este no tiene mucho sentido para nuestro problema a resolver, pero nos da una idea de la capacidad para extracción de características.

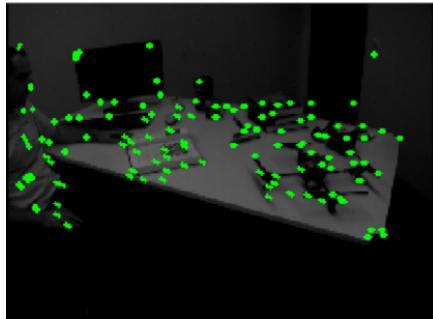


Figura 7.4: Comienzo de prueba Dynamic en el SVO



Figura 7.5: Seguimiento de prueba Dynamic en el SVO

### 7.1.2. Pruebas en interiores con un tablero de ajedrez con parámetros por defecto.

En estos vídeos se graban una serie de vídeos en mi habitación, con un tablero de ajedrez porque tiene una textura muy clara y bordes muy bien definidos, que serán muy útiles para encontrar características robustas para las que es fácil encontrar correspondencias entre fotogramas consecutivos, además de varios objetos colocados a su alrededor.

Lista de parámetros de configuración del algoritmo SVO

<b>trace_name</b>	“svo”	<b>structureoptim_max_pts</b>	20
<b>trace_dir</b>	“/tmp”	<b>structureoptim_num_iter</b>	5
<b>n_pyr_levels</b>	3	<b>loba_thresh</b>	2
<b>use_imu</b>	False	<b>loba_robust_huber_width</b>	1
<b>core_n_kfs</b>	3	<b>loba_num_iter</b>	0
<b>map_scale</b>	1	<b>kfselect_mindist</b>	0.12
<b>grid_size</b>	30	<b>triang_min_corner_score</b>	20
<b>init_min_disparity</b>	50	<b>triang_half_patch_size</b>	4
<b>init_min_tracked</b>	50	<b>subpix_n_iter</b>	10
<b>init_min_inliers</b>	40	<b>max_n_kfs</b>	10
<b>klt_max_level</b>	4	<b>img_imu_delay</b>	0
<b>klt_min_level</b>	2	<b>max_fts</b>	120
<b>reproj_thresh</b>	2	<b>quality_min_fts</b>	50
<b>poseoptim_thresh</b>	2	<b>quality_max_drop_fts</b>	40
<b>poseoptim_num_iter</b>	10		

Cuadro 7.1: Parámetros SVO

En la Tabla 7.1.2 se encuentran los distintos parámetros del sistema SVO, entre los que se encuentran los que indican el lugar donde se alma-

cena la información, la cantidad de características robustas necesarias para el seguimiento, la puntuación necesaria para identificar un punto como característica robusta o el número de puntos máximo que se optimiza en cada iteración.

En este vídeo se coloca un tablero de ajedrez encima de la cama con distintos objetos a su alrededor como un portátil, una botella y un libro. En las Figuras 7.6 y 7.7 se muestra la primera fase del algoritmo SVO, en la que se obtiene el mapa de triangulación con las trayectorias que siguen los puntos (características robustas) que se siguen a través de los fotogramas consecutivos, como se notifica en la salida de la terminal 7.8, donde vemos como con el seguimiento de 77 características robustas, siguiendo la restricción de la variable “init\_min\_tracked” de más de 50 características para formar el mapa de triangulación, y dejándolo construido una vez se superan los 50 píxeles de media en la desviación de estas características en el fotograma actual con respecto al primer fotograma.

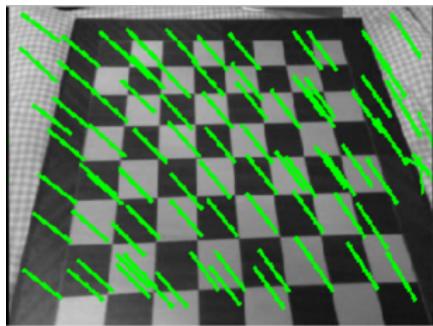


Figura 7.6: Seguimiento de características para inicializar mapa de triangulación de la prueba con tablero encima de la cama

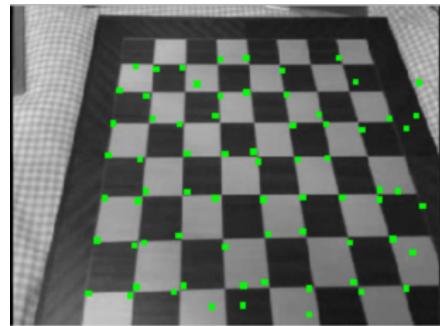


Figura 7.7: Primer fotograma con extracción de características de la prueba con tablero encima de la cama

```
[ INFO] [1626777191.448393485]: Init: KLT tracked 77 features
[ INFO] [1626777191.448474090]: Init: KLT 50.0146px average disparity.
[ INFO] [1626777191.449551593]: Init: Homography RANSAC 71 inliers.
[ INFO] [1626777191.451811085]: Init: Selected second frame, triangulated initial map.
```

Figura 7.8: Salida terminal triangulación del mapa de la prueba con tablero encima de la cama

En la Figura 5 podemos ver como el SVO notifica la pérdida de características suficientes para estimar la posición, intentando la relocalización, pero sin conseguirlo al no encontrar suficientes características que correspondan con otras de anteriores fotogramas. En el fotograma de la Figura

4 se observa como solo reconoce tres características robustas en la imagen para la localización.

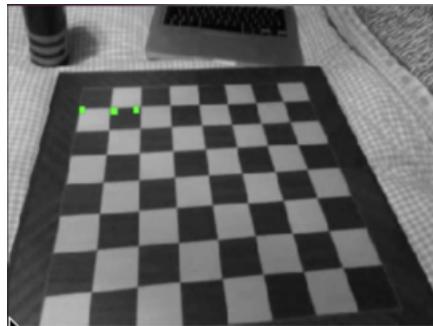


Figura 7.9: Extracción de características durante el vídeo de la prueba con tablero encima de la cama

```

WARN] [1626777655.732740635]: Relocalizing frame
INFO] [1626777655.765883186]: No reference keyframe.
INFO] [1626777655.832851203]: No reference keyframe.
WARN] [1626777655.866658192]: Not enough matched features.
INFO] [1626777655.934532335]: No reference keyframe.

```

Figura 7.10: Salida terminal durante el vídeo de la prueba con tablero encima de la cama

### 7.1.3. Pruebas en interiores con un tablero de ajedrez y con cambio de parámetros.

<code>grid_size</code>	10	<code>triang_min_corner_score</code>	5
<code>init_min_disparity</code>	10	<code>triang_half_patch_size</code>	15
<code>reproj_thresh</code>	10	<code>max_n_kfs</code>	0
<code>poseoptim_thresh</code>	5	<code>quality_min_fts</code>	20
<code>kfselect_mindist</code>	0.06		

Cuadro 7.2: Lista de parámetros SVO modificados

Desde esta sección en adelante se usarán estas modificaciones para cada una de las pruebas, al final de la sección [7.1.7](#) se comentará el por qué de la decisión.

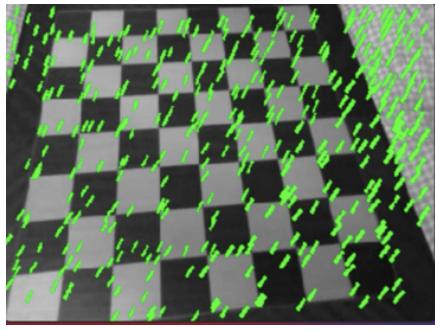


Figura 7.11: Seguimiento de características para inicializar mapa de triangulación en prueba en interior con tablero de ajedrez con nuevos parámetros

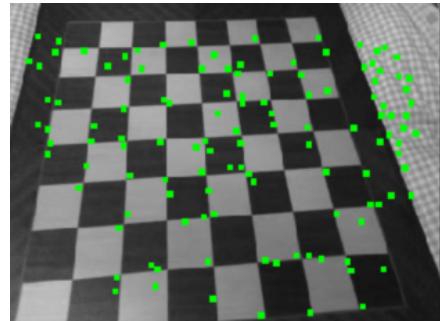


Figura 7.12: Primer fotograma con extracción de características en prueba en interior con tablero de ajedrez con nuevos parámetros

```
[ INFO] [1626781947.235338067]: Init: KLT tracked 377 features
[ INFO] [1626781947.235398096]: Init: KLT 9.62318px average disparity.
[ INFO] [1626781947.262978320]: Init: KLT tracked 375 features
[ INFO] [1626781947.263049223]: Init: KLT 10.6903px average disparity.
[ INFO] [1626781947.263674337]: Init: Homography RANSAC 372 inliers.
[ INFO] [1626781947.264008994]: Init: Selected second frame, triangulated initia
l map.
```

Figura 7.13: Salida terminal mapa triangulación en prueba en interior con tablero de ajedrez

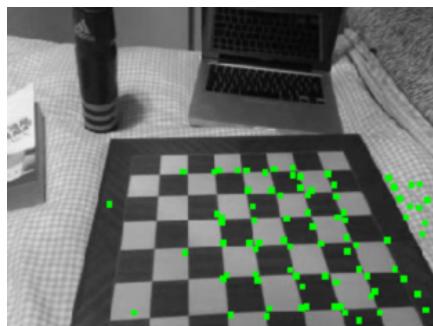


Figura 7.14: Extracción de características durante el vídeo en prueba en interior con tablero de ajedrez

En esta sección se muestran los resultados obtenidos del mismo vídeo de la subsección anterior, pero con los parámetros modificados, indicados de la Tabla 7.1.3. La cantidad de características robustas obtenidas son 375, y una vez la diferencia media entre el píxel de inicio y el píxel actual en el que se encuentra la característica robusta pase de 10 píxeles, se pasa a la segunda

fase del algoritmo. Donde como observamos en la Figura 7.14, se obtienen una gran cantidad de características robustas por fotograma, realizando así la correspondencia entre más puntos distintos.

#### 7.1.4. Pruebas en interiores con estampados extravagantes y con cambio de parámetros.

##### Prueba escritorio con tablero, ordenador y monitor

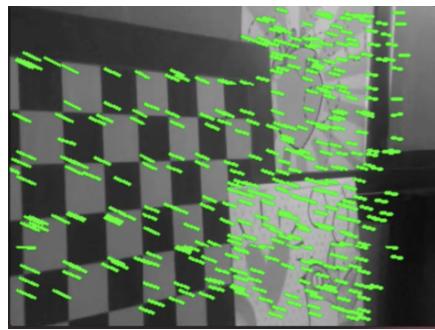


Figura 7.15: Seguimiento de características robustas para triangulación en prueba escritorio con tablero, ordenador y monitor

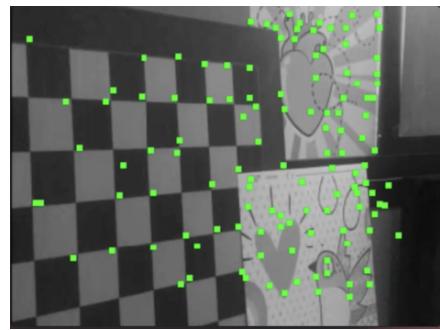


Figura 7.16: Primer fotograma con extracción de características en prueba escritorio con tablero, ordenador y monitor

```
[ INFO] [1626793685.873671319]: Init: KLT tracked 348 features  
[ INFO] [1626793685.873729900]: Init: KLT 10.0885px average disparity.  
[ INFO] [1626793685.874108907]: Init: Homography RANSAC 347 inliers.  
[ INFO] [1626793685.874270952]: Init: Selected second frame, triangulated initial map.
```

Figura 7.17: Salida terminal con el mapa de triangulación en prueba escritorio con tablero, ordenador y monitor

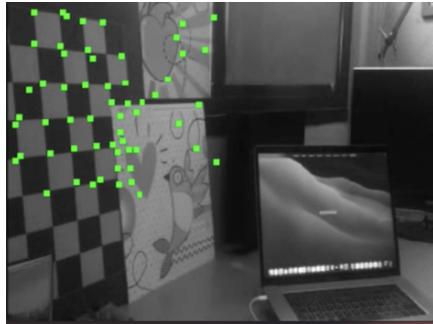


Figura 7.18: Extracción de características durante el vídeo en prueba escritorio con tablero, ordenador y monitor

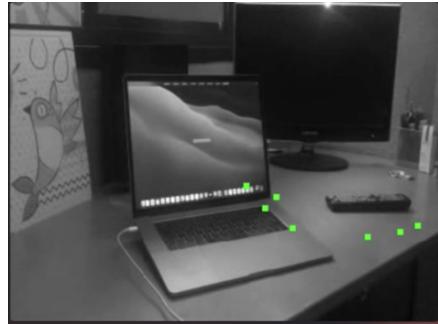


Figura 7.19: Extracción de características para relocalización de la escena en prueba escritorio con tablero, ordenador y monitor

```
[ WARN] [1626793709.190250721]: Not enough matched features.  
[ WARN] [1626793709.222903628]: Relocalizing frame
```

Figura 7.20: Salida terminal con mensaje de relocalización en prueba escritorio con tablero, ordenador y monitor

En este vídeo se colocan las distintas texturas y el tablero de ajedrez sobre la pared, con distintos objetos sobre la mesa como un portátil, un monitor y su mando. El vídeo empezará enfocando el tablero y las texturas, e irá alejándose de modo que aparezcan los objetos del escritorio.

Para la primera fase del SVO, los puntos característicos extraídos se reparten entre el tablero de ajedrez y las nuevas láminas, con estampados extraños de figuras y un alto contraste 7.15 7.16, llegando a las 348 características robustas antes de superar el mínimo de separación de 10 píxeles de media.

Conforme la imagen se aleja de esos puntos iniciales localizados por todo el tablero y estampados, en la segunda fase del algoritmo SVO, se comienza a perder correspondencias en las características, como se observa en la cantidad de puntos verdes en la 7.18. Una vez el número de correspondencias baja de las 20, como se indica en la variable de configuración “quality\_min\_fts”, salta el mensaje en terminal mostrado en la Figura 7.20 ante la falta de correspondencias suficientes, y SVO tiene que relocalizarse con la búsqueda de unos nuevos puntos como se ve en la Figura 7.19.

### Prueba con muñeco en escritorio

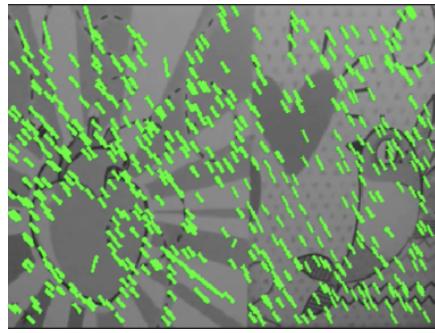


Figura 7.21: Seguimiento de características robustas para triangulación en prueba escritorio con muñeco

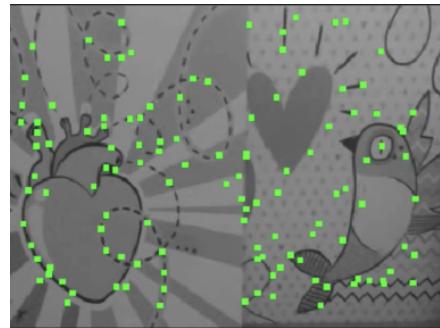


Figura 7.22: Primer fotograma con extracción de características en prueba escritorio con muñeco

```
[ INFO] [1626795161.732066762]: Init: KLT tracked 482 features
[ INFO] [1626795161.732132092]: Init: KLT 11.0158px average disparity.
[ INFO] [1626795161.732671810]: Init: Homography RANSAC 474 inliers.
[ INFO] [1626795161.736749370]: Init: Selected second frame, triangulated initial map.
```

Figura 7.23: Salida terminal con el mapa de triangulación en prueba escritorio con muñeco

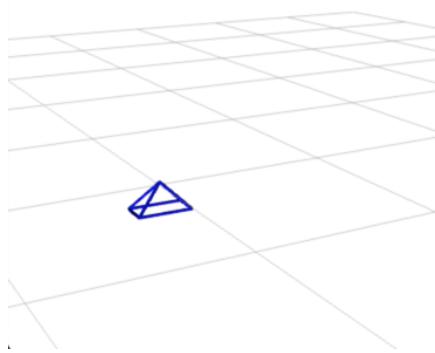


Figura 7.24: Inicio de estimación en prueba escritorio con muñeco

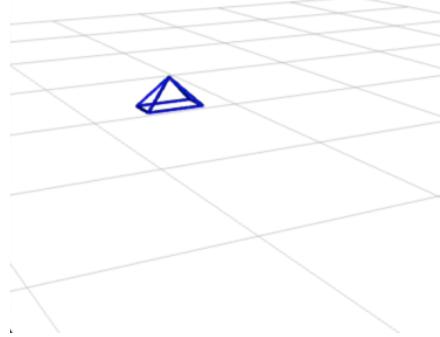


Figura 7.25: Estimación en fotogramas finales en la prueba escritorio con muñeco

Para este vídeo se colocan las texturas sobre la pared y un muñeco enfrente de ellos sobre la mesa. Empezando a grabar la escena enfocando solamente los estampados, para ir retrocediendo la cámara de posición hasta que salga en escena el muñeco en cuestión.

Observamos en las Figuras 7.21 y 7.23 como la cantidad de puntos evaluados es muy alta, 482 características robustas antes de superar la media de dispersión en más de 10 píxeles. Una vez sacado el mapa de triangulación, las características en los estampados son consistentes, como se puede ver en la Figura 7.22 y aún alejándonos de ellos se sigue manteniendo esa consistencia como vemos en la Figura 7.26.

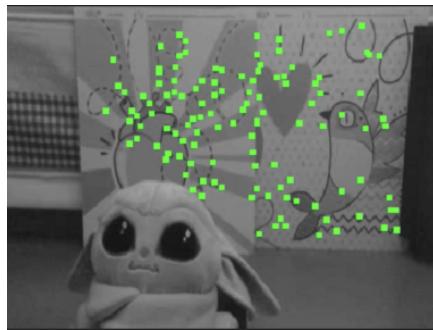


Figura 7.26: Extracción de características durante el vídeo en prueba escritorio con muñeco

Esto hace que la estimación de movimiento sobre el eje de profundidad sea correcta, comenzando en 7.24 y desplazándose hacia arriba hasta el final del vídeo, como se ve en la Figura 7.25.

#### 7.1.5. Pruebas en interiores con texturas extravagantes, resolución 720p y con cambio de parámetros.

##### Prueba escritorio con ordenador y monitor

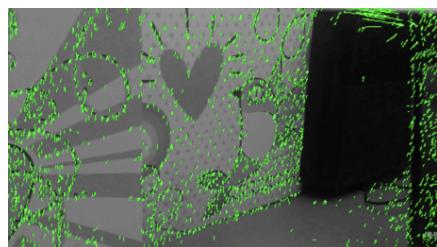


Figura 7.27: Seguimiento de características robustas para triangulación en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles

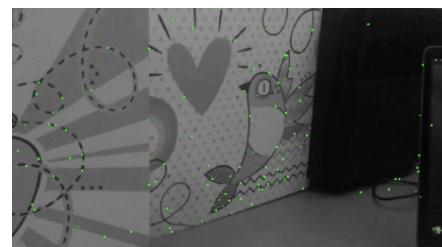


Figura 7.28: Primer fotograma con extracción de características en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles

```
[INFO] [1626797036.740601983]: Init: KLT tracked 2046 features
[INFO] [1626797036.741120567]: Init: KLT 16.0056px average disparity.
[INFO] [1626797036.743808149]: Init: Homography RANSAC 1913 inliers.
[INFO] [1626797036.749401458]: Init: Selected second frame, triangulated initial map.
```

Figura 7.29: Salida terminal con el mapa de triangulación en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles



Figura 7.30: Extracción de características durante el vídeo en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles

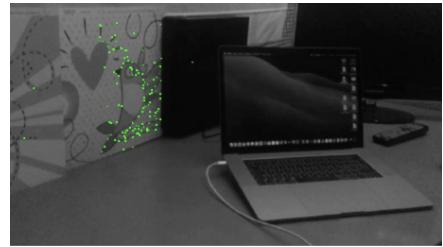


Figura 7.31: Extracción de características posterior en el vídeo de prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles

Este vídeo tiene una configuración similar al de la subsección 7.1.4, pero con la diferencia del tablero, que en este caso no aparece colocado en el entorno. Se encuentra grabado a la resolución original de 720p (1280x720 píxeles), y como se observa en la Figura 7.27 la cantidad de características seguidas para la triangulación del mapa llega a las 2046 para la primera fase del SVO.

La secuencia de movimiento entre las Figuras 7.28, 7.30 y 7.31, muestran como las características robustas seguidas para las correspondencias con el mapa de triangulación inicial, se concentran en la parte del estampado con forma de pájaro, ya que al existir una mayor resolución, es capaz de extraer características en zonas con contraste más alejadas de la imagen.

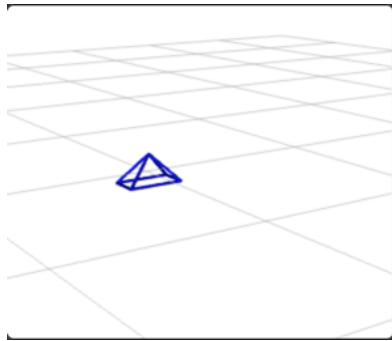


Figura 7.32: Inicio de estimación en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles

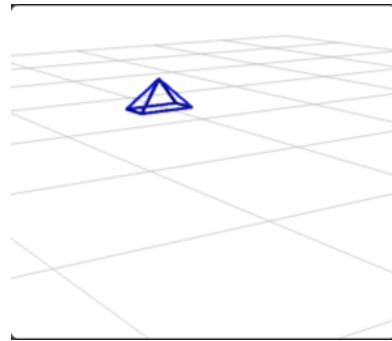


Figura 7.33: Estimación final en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles

Y como se muestra en las Figuras 7.32 y 7.33, la posición estimada de la cámara vemos como sigue detectando la traslación hacia atrás, pero no el giro dado para enfocar el resto de la mesa.

```
[WARN] [1626797065.814545378]: Lost 50 features!
[WARN] [1626797065.852678001]: Relocalizing frame
[WARN] [1626797065.859466344]: Not enough matched features.
[INFO] [1626797065.942474058]: No reference keyframe.
```

Figura 7.34: Salida terminal con mensaje de relocalización en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles

En la Figura 7.34 se muestra la salida de la terminal proporcionada por SVO, notificando la pérdida de características robustas, y dando la necesidad de la búsqueda de nuevos puntos para hacer las correspondencias 7.35. Una vez el algoritmo pierde la referencia, la posición estimada vuelve a ser la del punto inicial, tal y como se muestra en la Figura 7.36.

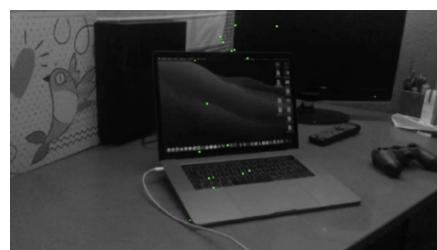


Figura 7.35: Búsqueda de nuevos puntos para relocalización en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles

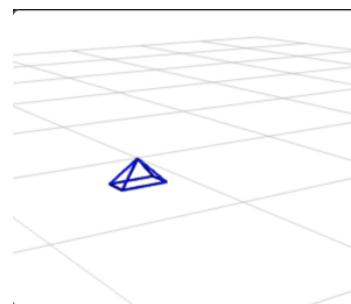


Figura 7.36: Pérdida de la estimación en prueba escritorio con ordenador y monitor a resolución 1280x720 píxeles

#### Prueba con objetos pequeños en escritorio

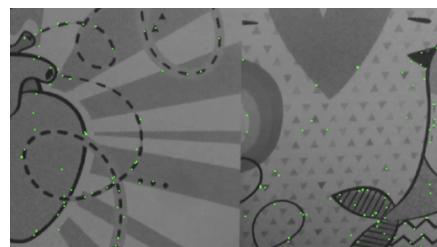


Figura 7.37: Primer fotograma con extracción de características en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles



Figura 7.38: Búsqueda de nuevos puntos para relocalización en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles



Figura 7.39: Seguimiento de características final en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles

Este vídeo tiene un movimiento similar al realizado en la subsección anterior 7.1.4, donde empezamos enfocando los estampados y nos vamos alejando, para apuntar a los distintos objetos colocados encima de la mesa.

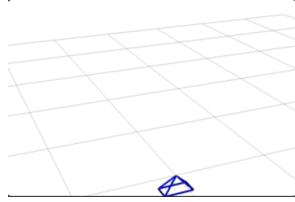


Figura 7.40: Inicio de estimación en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles

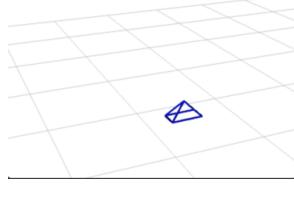


Figura 7.41:  
Seguimiento de la estimación en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles

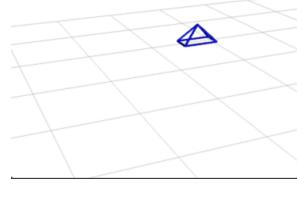


Figura 7.42:  
Finalización de la estimación en prueba escritorio con objetos pequeños a resolución 1280x720 píxeles

Las Figuras 7.37, 7.38 y 7.39 muestran fotogramas durante el transcurso del vídeo, que se corresponden con las posiciones estimadas de las Figuras 7.40, 7.41 y 7.42 respectivamente, conforme nos vamos alejando en el vídeo de las texturas, se pierde SVO en la localización de características, con la necesidad de buscar nuevos puntos para realizar las correspondencias con el mapa de triangulación y relocatearse con éxito, como vemos en la Figura 7.38, se mantienen los puntos para la estimación una vez nos alejamos 7.39.

Observamos cómo a pesar de tener una pérdida en el proceso de localización, es capaz de mantener una consistencia, y dar un movimiento alejándose del punto inicial en un eje, tal y como se hace en el vídeo.

### 7.1.6. Pruebas en interiores con Arucos/Graffiti y con cambio de parámetros.

#### Prueba 1 de cajas con distinta altura en el suelo

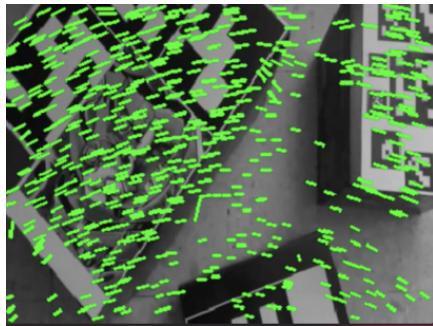


Figura 7.43: Seguimiento de características robustas para triangulación en prueba de cajas 1

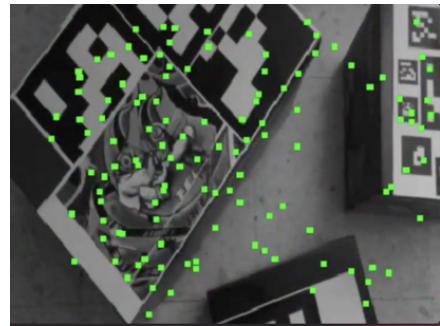


Figura 7.44: Primer fotograma con extracción de características en prueba de cajas 1

```
[ INFO] [1626800444.653523571]: Init: KLT tracked 564 features  
[ INFO] [1626800444.653754732]: Init: KLT 10.0275px average disparity.  
[ INFO] [1626800444.654452539]: Init: Homography RANSAC 558 inliers.  
[ INFO] [1626800444.654733589]: Init: Selected second frame, triangulated initia  
l map.
```

Figura 7.45: Salida terminal con el mapa de triangulación en prueba de cajas 1

En este vídeo se colocan tres cajas de distinta altura en el suelo, todas ellas con distintos estampados de Arucos, unos marcadores cuadrados compuestos por un amplio borde negro y una matriz binaria interior, que determina el color de la sección (blanco o negro), además de una imagen de graffiti, típica de los problemas de localización. Se vuelve a la resolución reducida de 346x260 píxeles original.

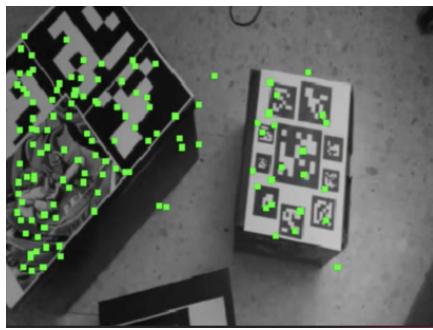


Figura 7.46: Seguimiento de características durante el vídeo de prueba de cajas 1

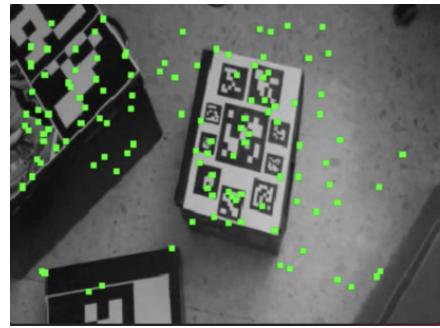


Figura 7.47: Seguimiento de características al desplazar la cámara a la derecha durante el vídeo de prueba de cajas 1

El mapa de triangulación se forma a partir de 564 características [7.43], al superar el límite de 10 píxeles de desviación media, y comienza la segunda fase del algoritmo SVO en la Figura [7.44]. Las puntos con correspondencia se localizan mayoritariamente en los bordes de los arucos y en distintas zonas del graffiti, patrón que se sigue viendo en la Figura [7.46], pero que, al desplazarnos con la cámara hacia la derecha, perdemos visión del graffiti y los distintos arucos de la caja grande, entonces el algoritmo busca nuevos puntos por el suelo, para realizar la correspondencia de características entre fotogramas distintos.

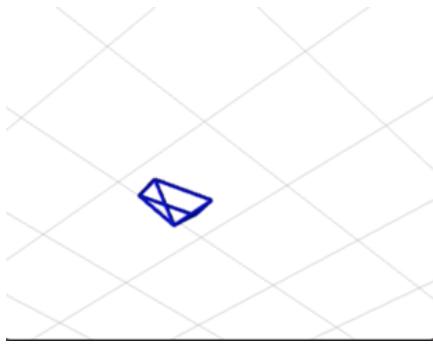


Figura 7.48: Inicio de la estimación en prueba de cajas 1

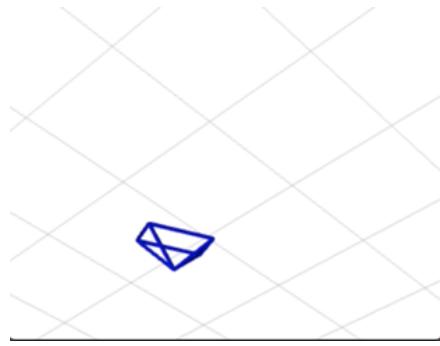


Figura 7.49: Final de la estimación en prueba de cajas 1

Finalmente, las Figuras [7.48] y [7.49] muestran la progresión de la estimación de la cámara, desde el momento del inicio al final del vídeo. Vemos como el movimiento de alejamiento de la cámara es capaz de estimarlo, e incluso el giro final se nota levemente, que debería corresponder con el enfoque de la cámara en la Figura [7.47], donde la cámara se desplaza ligeramente

a la derecha y se produce un giro para seguir enfocando a la caja grande.

### Prueba 2 de cajas con distinta altura en el suelo

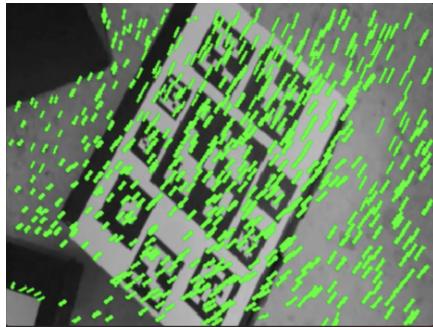


Figura 7.50: Seguimiento de características robustas para triangulación en prueba de cajas 2

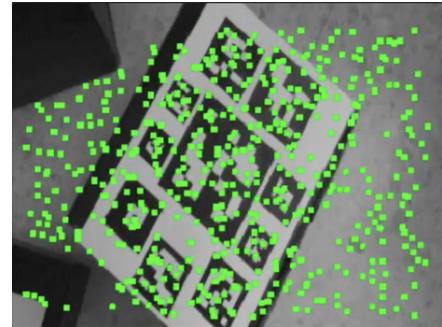


Figura 7.51: Primer fotograma con extracción de características en prueba de cajas 2

En este vídeo se vuelve a utilizar la misma configuración que en la subsección anterior 7.1.6, pero empezando a grabar la caja con arucos de distintos tamaños en vez de la caja más grande.

```
[ INFO] [1626800444.653523571]: Init: KLT tracked 564 features
[ INFO] [1626800444.653754732]: Init: KLT 10.0275px average disparity.
[ INFO] [1626800444.654452539]: Init: Homography RANSAC 558 inliers.
[ INFO] [1626800444.654733589]: Init: Selected second frame, triangulated initial map.
```

Figura 7.52: Salida terminal con el mapa de triangulación en prueba de cajas 2

El mapa de triangulación se genera con las características encontradas en los aruco de distinto tamaño, además de otras encontradas en el suelo de mármol 7.50, 553 características que se mantienen de forma consistente en el primer fotograma de la segunda fase 7.51, donde se comienza a realizar las correspondencias.

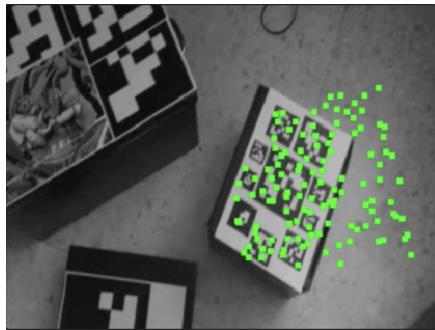


Figura 7.53: Extracción de características a la mitad de la prueba de cajas 2

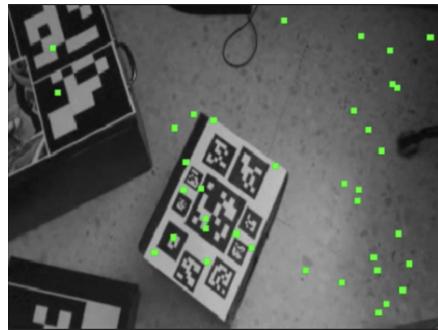


Figura 7.54: Extracción de características al final de la prueba de cajas 2

Al alejarnos de la caja y enfocar al resto, el SVO es capaz de mantener las características robustas de la caja inicial [7.53], pero una vez nos desplazamos a la derecha en el entorno, se empieza a seleccionar nuevas características para las correspondencias siguientes como observamos en la Figura [7.54].

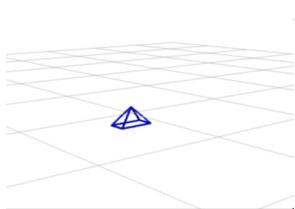


Figura 7.55: Inicio de la estimación en prueba de cajas 2

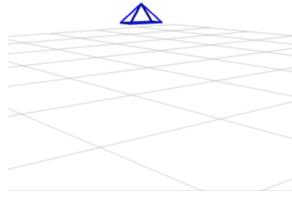


Figura 7.56:  
Estimación en la  
mitad de la prueba de  
cajas 2

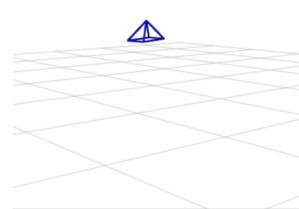


Figura 7.57: Final de  
la estimación en prue-  
ba de cajas 2

Los movimientos de la cámara estimados son similares a los ocurridos, ya que empezamos en el punto inicial [7.55], nos alejamos de la caja como vemos en la Figura [7.53], lo que lleva a la estimación de la Figura [7.56], pero finalmente el movimiento de desplazamiento hacia la derecha no queda registrado en la Figura [7.57], sino que estima una rotación del objetivo, como podemos ver entre la diferencia de ambas figuras en el prisma representativo de la cámara.

#### 7.1.7. Discusión de resultados

Las pruebas realizadas con el algoritmo SVO nos han ayudado a comprender mejor su funcionamiento, las restricciones del mismo y la utilidad que le podemos dar.

Es evidente que para que el algoritmo funcione correctamente, es necesario que la imagen contenga ciertos patrones con texturas que nos permitan extraer suficientes características robustas. En las pruebas se han usado desde un tablero de ajedrez hasta los típicos Arucos, usados en el estado del arte de forma común en los problemas de localización, pasando por estampados con figuras extravagantes. En definitiva, objetos o imágenes con un alto contraste, para que el SVO sea capaz de extraer las características robustas necesarias, con el objetivo de mantener un seguimiento entre fotogramas, realizando correspondencias entre los puntos.

El tablero de ajedrez, a pesar de dar características robustas, al ser un patrón repetitivo formado por filas de cuadrados alternando el color, puede dar problemas a la hora de realizar las correspondencias entre puntos de fotogramas distintos, dando lugar a estimaciones de posiciones incorrectas. Ocurre al contrario con los estampados de arucos junto con el graffiti y las láminas con figuras extravagantes, donde la consistencia en extracción de características y los patrones no tan repetitivos, los hacen muy útiles para la tarea de estimación de la posición.

Para facilitar la extracción de características, se hizo uso de la recomendación del usuario MortenBeier de GitHub<sup>1</sup>, siendo menos estrictos a la hora de evaluar cada píxel y emplearlo para la correspondencia de características robustas. Por lo tanto, se mantendrían una vez cambiados y comprobado su eficacia en los resultados, perdiendo en menos ocasiones el seguimiento para la estimación de la cámara.

Al limitar la resolución de los vídeos en un inicio a 346x260 píxeles, uno podría pensar que aumentando la resolución sería capaz de mejorar los resultados de la estimación, aumentando el número de características encontradas y una mejor estimación de la cámara. La realidad es que se siguen necesitando de altos contrastes en la imagen para su correcto funcionamiento, y puede llegar a seleccionar características poco perceptibles, que se irán perdiendo con mayor facilidad una vez se aleja la cámara de ellas. Por ello la resolución no es un factor determinante a la hora de la estimación de la cámara, siempre y cuando se tengan referencias lo suficientemente claras en la imagen.

En todas las pruebas realizadas y las mostradas en la memoria, los movimientos a realizar con la cámara deben ser suaves para la extracción de características y posterior uso para las correspondencias, para minimizar así el riesgo de pérdida en el proceso. Además según el propio repositorio de SVO<sup>2</sup>, la estimación funciona en movimientos hacia atrás, ya que en movimientos hacia delante no funciona correctamente (ya que necesita estimaciones de profundidad positivas).

<sup>1</sup>[https://github.com/uzh-rpg/rpg\\_svo/issues/205](https://github.com/uzh-rpg/rpg_svo/issues/205)

<sup>2</sup>[https://github.com/uzh-rpg/rpg\\_svo/wiki/Obtaining-Best-Performance](https://github.com/uzh-rpg/rpg_svo/wiki/Obtaining-Best-Performance)

Finalmente, tras las pruebas realizadas, sobre todo las dos últimas mostradas con cajas sobre el suelo, se ha comprobado como SVO no es capaz de detectar los movimientos de traslación de la cámara hacia los lados, interpretándolos a veces como una rotación, o un acercamiento de la cámara a la posición inicial para después volver a alejarse. En cualquier caso, nunca ha sido capaz de estimar este tipo de movimientos tras varias pruebas en distintos vídeos.

## 7.2. V2E

Para medir las ventajas proporcionadas por el uso de eventos, en este caso uso de eventos simulados por V2E, se medirán los tamaños que ocupan los eventos, y se compararán con la cantidad de espacio que necesitan los archivos con imágenes en escala de grises y en formato RGB. También observando los cambios de eventos en un mismo vídeo con distintas resoluciones, o el umbral seleccionado para generar un evento en el simulador. Además, al estar usando un simulador como lo es V2E, se medirán los tiempos que tarda para simular los eventos de un vídeo en distintas resoluciones, o el tiempo empleado con el uso de la aceleración por GPU CUDA frente al que necesita usando solamente la potencia de la CPU.

### 7.2.1. Tamaño

#### Tamaño de eventos generados en un mismo vídeo con distinto umbral

	Eventos umbral 0.3	Tamaño umbral 0.3 (MB)	Eventos umbral 0.5	Tamaño umbral 0.5 (MB)	Umbral 0.3	Umbral 0.5
Airground	59.540.000	1.595,65	26.550.000	711,28	738,16	329,16
Prueba con muñeco en escritorio	46.140.000	1.256,88	24.720.000	673,71	572,03	306,47

Cuadro 7.3: Tamaño de archivo con distinto umbral en la generación de eventos

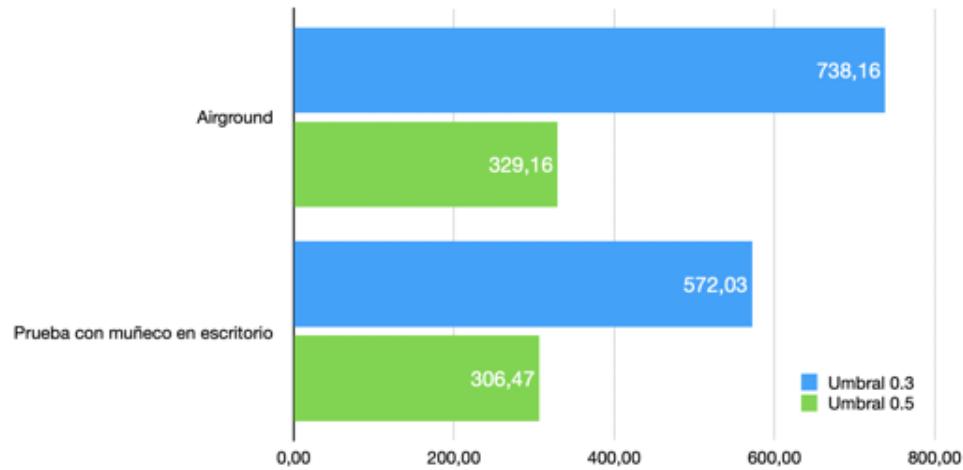


Figura 7.58: Gráfica tamaño requerido para almacenar los eventos según umbrales distintos en MB

En la Tabla 7.2.1 se muestran en las columnas 1 y 3 la cantidad de eventos generados dependiendo del umbral, las columnas 2 y 4 el tamaño del archivo “v2e-dvs-events.txt” generado por V2E, y las columnas 5 y 6 muestran el tamaño necesario para almacenar todos los eventos generados, representadas en la Figura 7.58, se obtiene como el tamaño de cada evento son 2 byte \* 2 (x, y) + 4byte \* 2 (sec, nsecs) + 1 byte (polaridad) que equivalen a 13 bytes por evento.

#### Tamaño distintas resoluciones mismo vídeo

	Eventos umbral 0.3	Tamaño umbral 0.3 (MB)	Eventos umbral 0.5	Tamaño umbral 0.5 (MB)	Umbral 0.3	Umbral 0.5
Airground	59.540.000	1.595,65	26.550.000	711,28	738,16	329,16
Prueba con muñeco en escritorio	46.140.000	1.256,88	24.720.000	673,71	572,03	306,47

Cuadro 7.4: Tamaño de archivo con distinta resolución en la generación de eventos

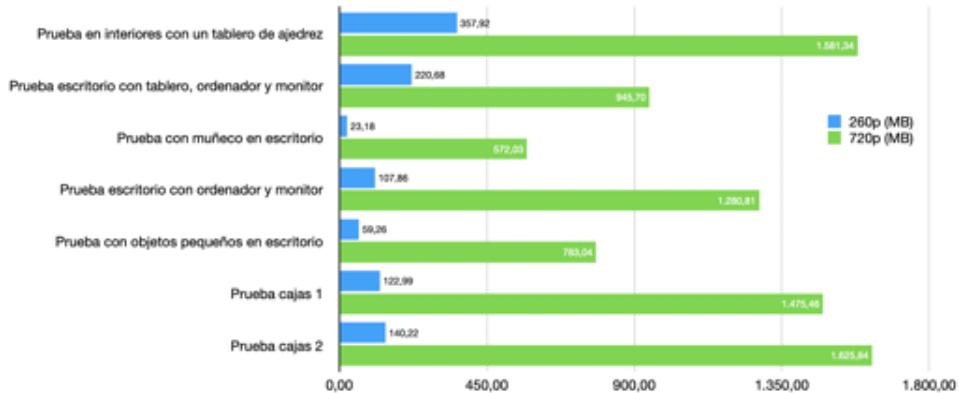


Figura 7.59: Gráfica tamaño requerido para almacenar los eventos según distintas resoluciones en MB

En la Tabla 7.2.1 sigue el mismo esqueleto que la Tabla 7.2.1, con la representación del tamaño necesario para los eventos de un mismo vídeo en distintas resoluciones (720p y 346x260p) en la Figura 7.59, donde se muestran en un gráfico de barras las diferencias de tamaños necesarios para almacenar los eventos de un mismo vídeo, en resolución 346x260 píxeles y en 1280x720 píxeles. Esta información nos será útil para visualizar las ventajas del uso de eventos frente al uso de imágenes tradicionales en ciertas tareas de visión por computador.

Para observar la cantidad de píxeles por cada evento de media, se hace uso de los valores mostrados en la Tabla 7.2.1, donde las dos primeras columnas muestran la cantidad de fotogramas que contiene el vídeo originalmente y tras pasar por la red Super-SloMo, el proceso intermedio encargado de generar los fotogramas intermedios usados exclusivamente para simular los eventos.

	Frames con SloMo	Frames normales	Eventos/frame SloMo 260p	Eventos/frame SloMo 720p	Gris/frame 260p	Gris/frame 720p	Proporción 260p	Proporción 720p
Prueba en interiores con un tablero de ajedrez	8.352	697	3.456,7	15.271,8	89.960	2.764.800	26,03	181,04
Prueba escritorio con tablero, ordenador y monitor	10.524	878	1.691,4	7.248,2	89.960	2.764.800	53,19	381,45
Prueba con muñeco en escritorio	10.212	852	183,1	4.518,2	89.960	2.764.800	491,27	611,92
Prueba escritorio con ordenador y monitor	11.604	968	749,7	8.903,0	89.960	2.764.800	119,99	310,55
Prueba con objetos pequeños en escritorio	11.568	965	413,2	5.459,9	89.960	2.764.800	217,71	506,38
Prueba cajas 1	7.092	592	1.398,8	16.780,9	89.960	2.764.800	64,31	164,76
Prueba cajas 2	9.612	802	1.176,7	13.643,4	89.960	2.764.800	76,45	202,65

El cálculo de los eventos generados por cada fotograma de media, se encuentra en las dos siguientes columnas (una para cada resolución), seguido de la cantidad de píxeles que contiene cada fotograma por resolución.

Las últimas dos columnas quedan representadas en la gráfica de la Figura 7.60, mostrando la cantidad de píxeles por evento generado que hay de media en cada uno de los vídeos con distintas resoluciones. Apoyándonos en estos valores, podremos valorar en la siguiente sección, si esta reducción de datos implica también un menor tamaño en bytes, y por ello, una ventaja computacional al tratar la información.

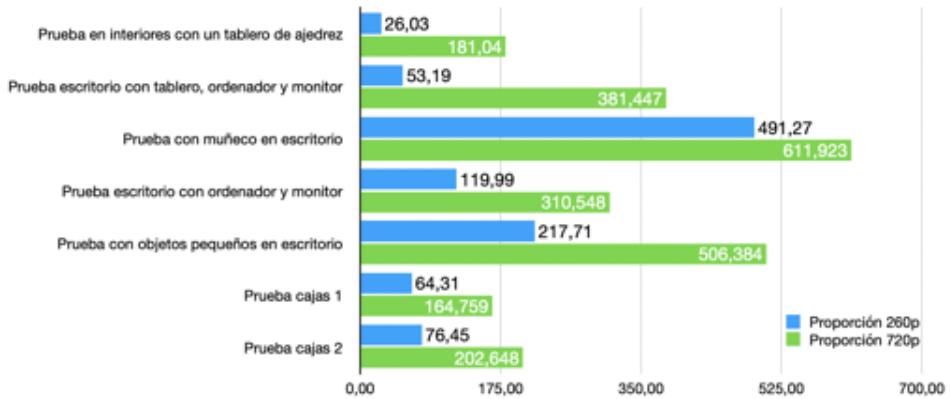


Figura 7.60: Gráfica con la proporción de píxeles de media por evento en cada vídeo

### Comparativa y discusión Imagen RGB vs escala de grises vs Eventos

Como hemos estudiado en la subsección anterior 7.2.1, cada evento necesita de 13 bytes para su representación. Cada píxel puede tener una representación en escala de grises o en RGB, la diferencia está en representarlo con 256 valores, o lo que es lo mismo, 8 bits que equivalen a 1 byte, o representarlo con 256 para cada uno de los colores, rojo, verde y azul, dando lugar a 3 bytes por píxel.

	Proporción 260p	Proporción 720p	Tamaño eventos (bytes)	Tamaño Gris 260p (bytes)	Tamaño RGB 260p (bytes)	Tamaño Gris 720p (bytes)	Tamaño RGB 720p (bytes)
Prueba en interiores con un tablero de ajedrez	26,03	181,04	13	26	78	181	543
Prueba escritorio con tablero, ordenador y monitor	53,19	381,45	13	53	160	381	1144
Prueba con muñeco en escritorio	491,27	611,92	13	491	1474	612	1836
Prueba escritorio con ordenador y monitor	119,99	310,55	13	120	360	311	932
Prueba con objetos pequeños en escritorio	217,71	506,38	13	218	653	506	1519
Prueba cajas 1	64,31	164,76	13	64	193	165	494
Prueba cajas 2	76,45	202,65	13	76	229	203	608

Cuadro 7.5: Tamaño requerido para almacenar la información equivalente a un evento en cada prueba

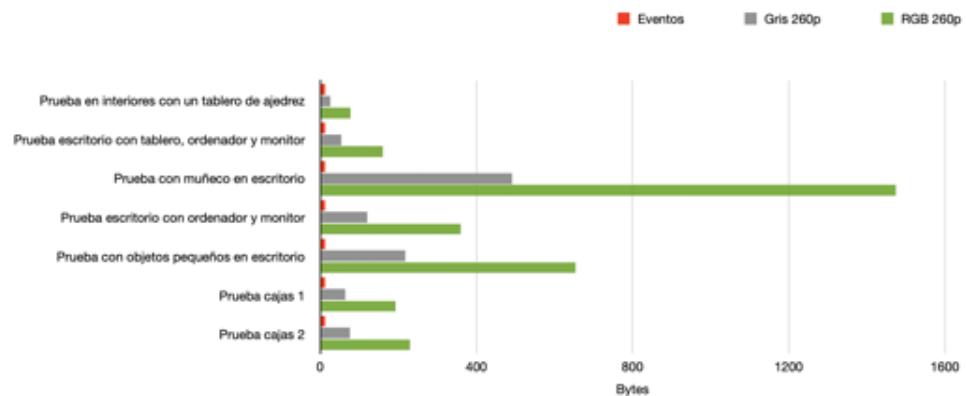


Figura 7.61: Gráfica con los tamaños equivalentes a un evento en cada vídeo según el formato empleado

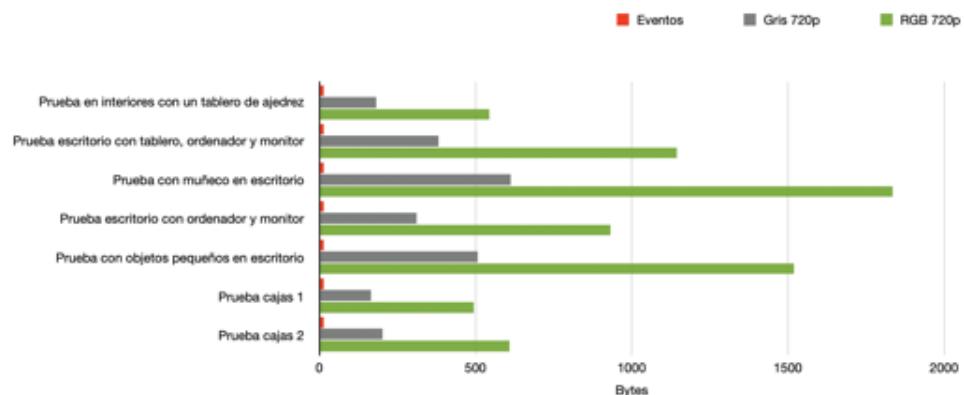


Figura 7.62: Gráfica con los tamaños equivalentes a un evento en cada vídeo según el formato empleado

En los ejemplos probados, los valores son determinantes en cuanto a los tamaños de los archivos dependiendo del formato en el que se guarde la información. Según vemos en las Figuras 7.61 y 7.62, la información equivalente a un evento (13 bytes) en cada uno de los vídeos, requiere de una gran mayor cantidad de almacenamiento.

Los porcentajes del tamaño de información usado por los eventos depende directamente del número de eventos captados durante el vídeo, en las pruebas con mayor densidad de objetos y mayores cambios en la intensidad de brillo entre píxeles (mayor contraste), resultan en un número de eventos total más grande y son aquellas donde las barras de las gráficas son más bajas, indican menor ventaja en reducción de tamaño al usar eventos frente a píxeles en formato RGB o en escala de grises.

Estos porcentajes rondan entre un 10-25 % de eventos por cada fotograma en escala de grises con resolución 346x260 píxeles en las pruebas. Y solamente se ahorra la mitad (49,95 %) del tamaño en la prueba en interiores con un tablero de ajedrez al usar eventos frente a píxeles en escala de grises. Pero si nos fijamos en las pruebas con muñeco en escritorio y en la prueba con objetos pequeños en escritorio, las cuales tienen la mayor proporción de píxeles por eventos generados como se indica en la Tabla 7.2.1, solo necesitan un 0.5-5 % del total de información almacenado por los píxeles (sean en formato RGB o escala de grises), suponiendo un grandísimo ahorro en el tamaño de la información para su tratamiento.

Es evidente que conforme más alta sea la resolución del vídeo, al ser la proporción de píxeles por evento generado más grande, implica una mayor diferencia, teniendo menor carga al tratar la información con el uso de eventos, frente a la carga con el tratamiento de imágenes tradicionales.

### 7.2.2. Tiempos de ejecución

Otro de los factores a tener en cuenta es el tiempo que se tarda en generar los eventos, comparando lo que requiere el sistema en convertir vídeos en distintas resoluciones a eventos. Además, comprobar los beneficios de usar la potencia de una GPU frente al uso de la CPU.

#### Aceleración GPU CUDA vs Ordenador CPU

El sistema V2E tiene limitación en cuanto a la carga computacional en caso de no usar CUDA<sup>3</sup>, siendo solo posible la generación de eventos en resoluciones pequeñas como la recomendada por los autores de 346x260 píxeles. Por ello, las comparativas de tiempos entre el uso de la GPU por el servidor y la CPU de mi ordenador personal, vendrán dadas en vídeos con la resolución 346x260 píxeles.

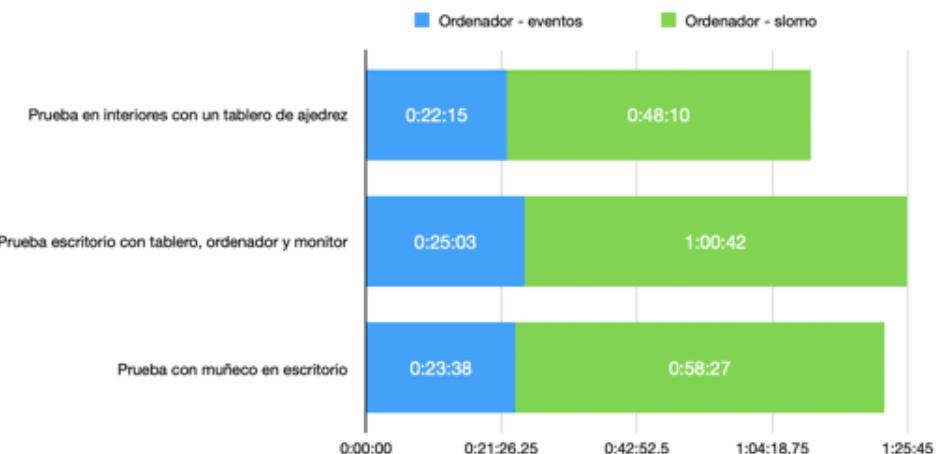


Figura 7.63: Tiempo empleado en generar eventos de un vídeo haciendo uso de la CPU

<sup>3</sup><https://developer.nvidia.com/cuda-zone>

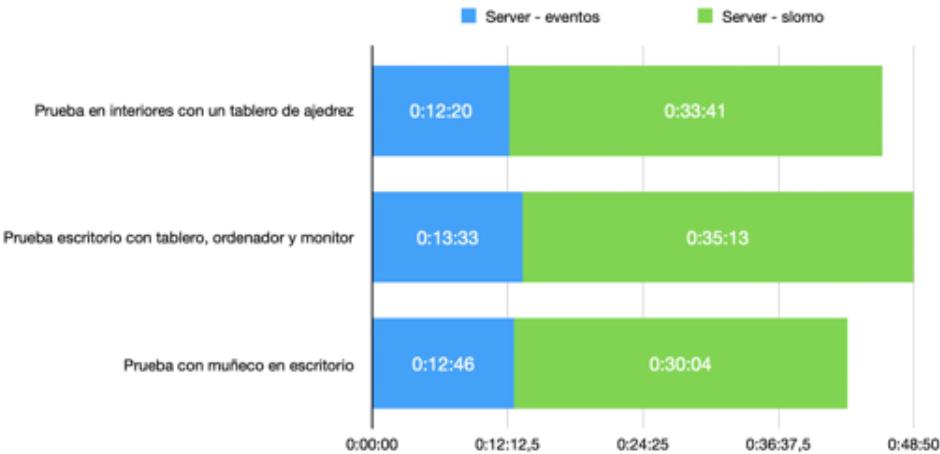


Figura 7.64: Tiempo empleado en generar eventos de un vídeo con uso de GPU

En la Figuras 7.63 y 7.64 se muestran los tiempos de tres conversiones de vídeos a eventos, la primera de las figuras con la CPU, y la segunda aprovechando la potencia de la GPU del servidor. Estos vídeos de prueba tienen una duración total que varía en un intervalo de 5 segundos, por lo tanto nos sirve para compararlos. Como vemos las diferencias de tiempo entre las pruebas con respecto a la fase final, donde se generan los eventos, es mínima, variando en pocos minutos, a diferencia de la fase en la que la red Super-SloMo interpola los fotogramas del vídeo, donde podemos observar una gran diferencia en el tiempo requerido para la tarea. Con el uso de la CPU, la diferencia entre tiempos para realizar la interpolación llega a los 12 minutos entre vídeos, en este caso el primer vídeo tiene una duración de 23 segundos, mientras que el segundo de ellos dura 28 segundos. Esta diferencia es lógica que provoque una demora en el tiempo de ejecución, al tener una mayor cantidad de fotogramas a interpolar, pero si observamos la diferencia producida con el uso de una GPU, la diferencia baja de 12 minutos a un apenas minuto y medio.

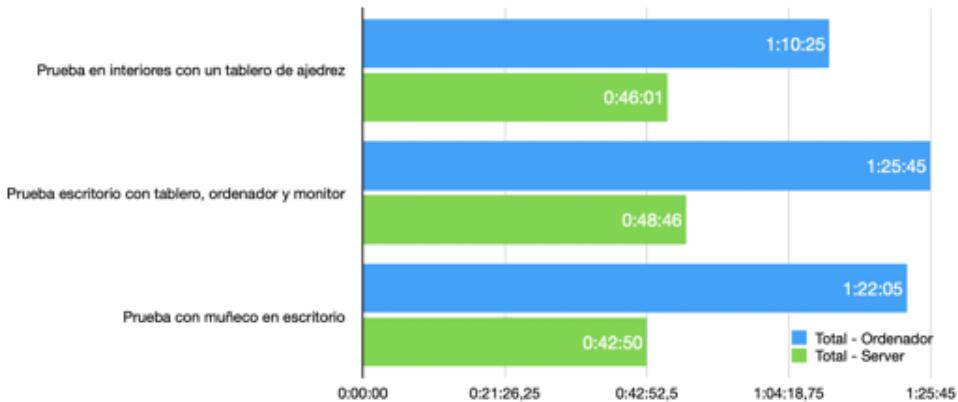


Figura 7.65: Tiempo total empleado por V2E haciendo uso de la CPU (azul) o de la GPU (verde)

Las proporciones de tiempo necesitado entre las dos distintas fases se mantiene a lo largo de los vídeos, se haga uso o no de la GPU. Pero el uso de la GPU es determinante, ya que la fase de interpolación tiene unas bajadas de tiempo de entre 15 a 30 minutos, y la fase de simulación de eventos disminuye unos 10 minutos en cada una de las pruebas. Esto conlleva una bajada de tiempos totales considerable en cada una de las pruebas, como se puede observar en la Figura 7.65.

### Distintas resoluciones 720p vs 346x260p

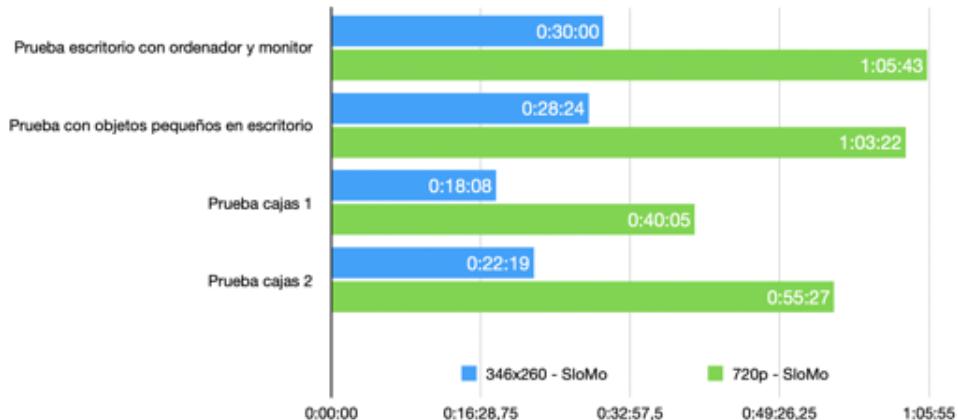


Figura 7.66: Tiempos de la fase de interpolación en distintas resoluciones

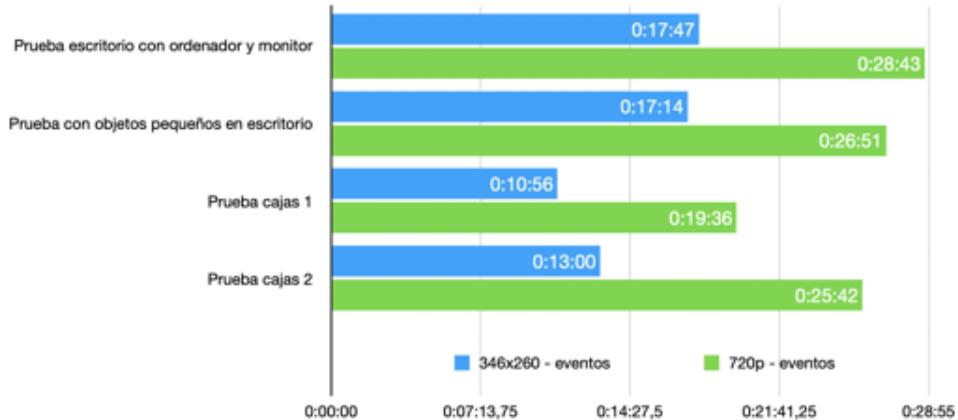


Figura 7.67: Tiempos de la fase de simulación de eventos en distintas resoluciones

Siguiendo la misma estructura que la sección anterior, se van a comparar los tiempos producidos en la generación en distintas resoluciones de un mismo vídeo. En este caso utilizamos dos parejas de vídeos con duración similar, como se pueden observar en los resultados de las Figuras 7.66 y 7.67, donde los tiempos requeridos son similares entre las parejas de pruebas. Como se podía intuir, los tiempos requeridos para una resolución más alta requiere de más tiempo, tardando el doble a 720p que en 346x260 píxeles de resolución, tanto en la fase de interpolación como en la de simulación de eventos. Por lo tanto, los tiempos en total de los vídeos en resolución 720p serán el doble que los de menor resolución como se observa en la Figura 7.68.

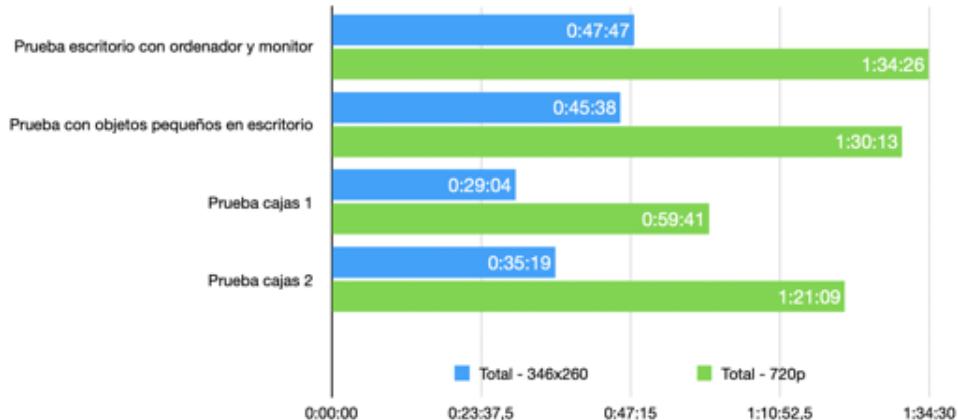


Figura 7.68: Tiempos totales del simulador V2E con distintas resoluciones

En definitiva, el uso de eventos propone una importante reducción en el tamaño de la información a tratar, consiguiendo reducir en ocasiones el

tamaño de la información hasta 100 veces menos, ofreciendo la posibilidad de usar técnicas de visión por computador con menor carga computacional y una más rápida transmisión de los datos del sensor.

### 7.3. Reconstrucción 3D

Finalmente tenemos el componente EMVS encargado de la reconstrucción 3D, listo para usar con cada una de las pruebas propias realizadas, una vez extraídos los eventos con V2E, estimadas las posiciones de la cámara y unidas a un mismo fichero binario con el paquete Crear\_rosbag, donde además se incluyen los parámetros de calibración de la cámara del iPhone<sup>4</sup> Figura 7.69 y 7.70.

```
1 cam_model: ATAN
2 cam_width: 346
3 cam_height: 260
4 cam_fx: 2090.4
5 cam_fy: 2094.4
6 cam_cx: 173
7 cam_cy: 130
```

Figura 7.69: Calibración en vídeos a 346x260 píxeles grabados con el iPhone SE

```
1 cam_model: ATAN
2 cam_width: 1280
3 cam_height: 720
4 cam_fx: 7733.3333
5 cam_fy: 5800.0
6 cam_cx: 640.0
7 cam_cy: 360.0
```

Figura 7.70: Calibración en vídeos a 720p grabados con el iPhone SE

Es importante tener unos parámetros de calibración correctos para la reconstrucción 3D, ya que tiene un efecto directo en los resultados, como podemos observar en la Figura 7.71 y 7.72, donde el vídeo queda descuadrado por distintos errores en los parámetros de calibración de la cámara.

<sup>4</sup><https://www.devicespecifications.com/en/model/7a423ad7>

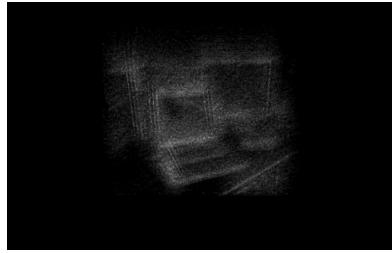


Figura 7.71: Ejemplo de malos parámetros de calibración con respecto a la resolución del vídeo

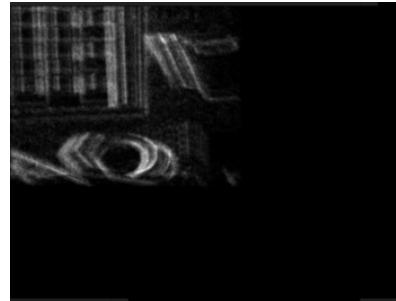


Figura 7.72: Ejemplo de malos parámetros de calibración con respecto al punto central de la cámara

### 7.3.1. Ejemplos propios

#### Prueba con muñeco en escritorio

En esta prueba grabamos un entorno con un muñeco encima de la mesa de escritorio, sin ningún otro objeto a su alrededor. Para situar el vídeo en un intervalo donde la reconstrucción 3D represente el entorno con el muñeco, se escogen los parámetros de configuración de la Tabla 7.3.1.

Parámetros de configuración para el EMVS	
<b>min_depth (metros)</b>	0.7
<b>max_depth (metros)</b>	0.75
<b>start_time_s (segundos)</b>	25.0
<b>stop_time_s (segundos)</b>	25.1
<b>dimZ (capas)</b>	5

Cuadro 7.6: Parámetros de configuración del EMVS para la prueba con muñeco en escritorio



Figura 7.73: Fotograma de la prueba con muñeco en escritorio, en el intervalo indicado por los parámetros “start\_time\_s” y “stop\_time\_s” indicados en la Tabla 7.3.1

En la Figura 7.73 se muestra un fotograma representativo con la escena sacada de los tiempos marcados en los parámetros de configuración, que dan como resultado las imágenes generadas con EMVS, mostrando los eventos recogidos para formar el espacio de disparidad Figura 75, un mapa de confianza, donde se muestran con mayor brillo los eventos en los que el EMVS tiene una mayor confianza en su estimación sobre profundidad Figura 76, y finalmente, el mapa de profundidad de la imagen en escala de grises Figura 77, y a color en la Figura 78, con el número de colores equivalente al parámetro de profundidad dado en la variable “dimZ” de la Tabla 7.3.1.

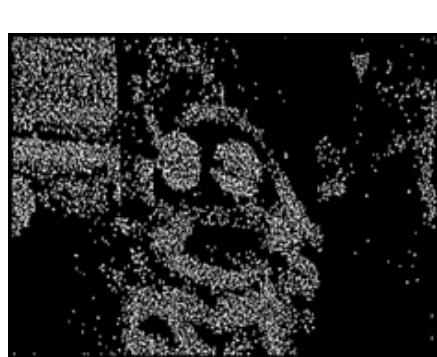


Figura 7.74: Eventos recogidos para la reconstrucción 3D de la prueba con muñeco en escritorio



Figura 7.75: Mapa de confianza del reconstructor EMVS en la estimación de la profundidad de cada evento de la prueba con muñeco en escritorio

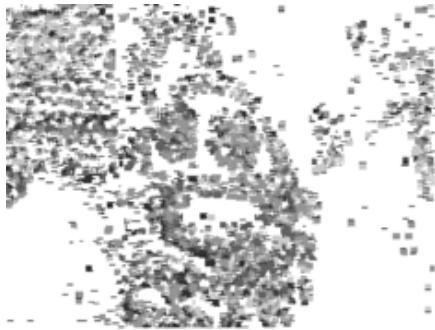


Figura 7.76: Mapa de profundidad en escala de grises de la prueba con muñeco en escritorio

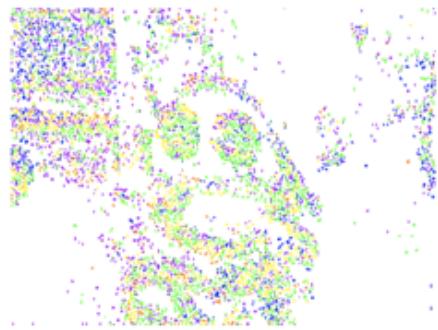


Figura 7.77: Mapa de profundidad a color de la prueba con muñeco en escritorio

Con el mapa de profundidad a color podemos observar como los eventos encontrados en la figura del muñeco suelen estar de color verde, amarillo o naranja, colores respectivos a las tres capas de profundidad más próximas a la cámara. Observamos también, como en los estampados de la pared donde se han generado eventos suelen estar compuestos por colores más fríos (azul y morado), al igual que en la cortina, que representan las dos capas más profundas o alejadas de la cámara. En los eventos situados al borde del muñeco o de la cortina, se ve como el algoritmo tiene dificultades para estimar correctamente la profundidad. En los bordes del muñeco se ve cómo varían los eventos entre las distintas capas, al igual que en el borde de la cortina, donde se encuentran eventos con capas de profundidad más cercanas a la cámara, cuando tendrían que tener la misma profundidad los eventos pertenecientes a la cortina. Esto sucede por culpa del ruido en la captación de eventos, además que los errores en la estimación de profundidad en algunos de los eventos es principalmente causa de la insuficiente precisión en la estimación de la posición de la cámara.

Esta diferencia de capas queda más evidente si disminuimos el número de capas a 3, cambiando el valor de la variable “dimZ”, donde el mapa de profundidad Figura 7.78, muestra al muñeco acaparando la mayor parte de la dos capas de profundidad más cercanas, y el resto se concentra en la capa más profunda, de color morado.



Figura 7.78: Mapa de profundidad a color de la prueba con muñeco en escritorio con 3 capas de profundidad

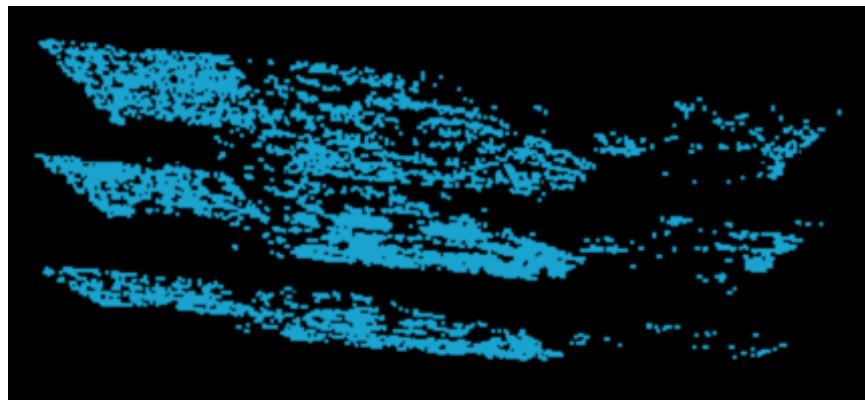


Figura 7.79: Nube de puntos generada con 3 capas de profundidad de la prueba con muñeco en escritorio

Con la nube de puntos generada [7.79](#), se pueden apreciar las 3 distintas capas de profundidad estimadas, siendo la capa de abajo la más cercana a la cámara. En la parte izquierda de la imagen, se encuentra la zona de la cortina con la pared, donde como se observa en la nube de puntos, existen eventos de esa zona en las tres distintas capas de profundidad, pero es la última capa la que tiene mayor densidad de eventos en esa zona.

También ocurre en la parte derecha de la imagen, donde se generan pocos eventos, correspondientes al estampado extravagante del pez-pájaro y al objeto negro delante de él situado al borde de la imagen, vemos que la estimación de los eventos de esa zona se centra en las dos últimas capas de profundidad.

Por último, la parte central de la imagen corresponde a la del muñeco, como comentamos anteriormente, los bordes del muñeco obtienen una estimación de profundidad pertenecientes a la última capa, pero el grueso de

eventos relacionados con el muñeco, referentes al cuerpo y ojos, se concientran en su mayoría en las dos capas más cercanas a la cámara.

### Prueba con muñeco en escritorio a 720p

Esta prueba usa el vídeo de la sección anterior [7.3.1] pero con una resolución de 720p, esto implica una mayor cantidad de eventos para la reconstrucción 3D. Al cambiar de resolución, a la lista de parámetros hay que añadir las dimensiones en x e y con las variables “dimX” y “dimY”, dándole los valores 1280 y 720 respectivamente. Por lo demás se mantienen los mismos parámetros del anterior experimento.

La cantidad de eventos a estimar la profundidad se muestran en la Figura [7.80], con una gran densidad de los mismos en cada uno de los contrastes presentes en la imagen.



Figura 7.80: Eventos recogidos para la reconstrucción 3D de la prueba con muñeco en escritorio a una resolución de 1280x720 píxeles

La estimación de profundidad [7.81], donde todas las zonas donde se encuentren los eventos obtienen una estimación de todas las distintas capas de profundidad.

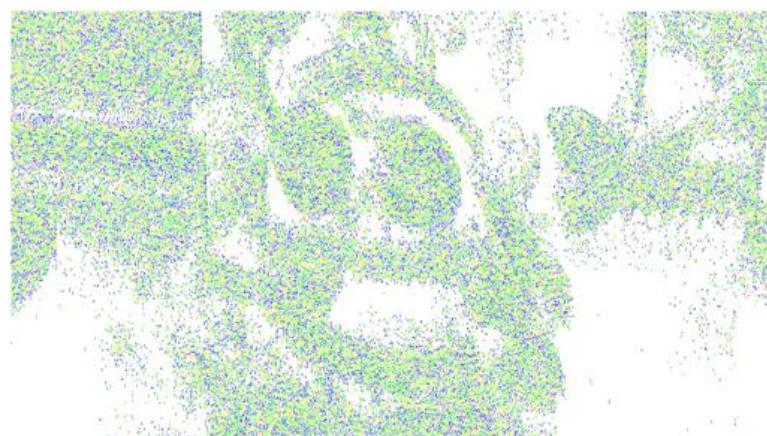


Figura 7.81: Mapa de profundidad a color de la prueba con muñeco en escritorio a una resolución de 1280x720 píxeles

#### Prueba con objetos pequeños en escritorio

Para esta prueba se usa un vídeo donde aparecen una cajita y mitad de una pelota pequeña en lugar del muñeco, en la misma mesa de escritorio con los mismo fondos de estampados [7.82]



Figura 7.82: Fotograma de la prueba con objetos pequeños en escritorio, en el intervalo indicado por los parámetros “start\_time\_s” y “stop\_time\_s” de la Tabla 7.3.1

Parámetros de configuración para el EMVS	
<b>min_depth (metros)</b>	0.7
<b>max_depth (metros)</b>	0.75
<b>start_time_s (segundos)</b>	15.0
<b>stop_time_s (segundos)</b>	16.0
<b>dimZ (capas)</b>	2

Cuadro 7.7: Parámetros de configuración del EMVS para la prueba con objetos pequeños en escritorio

Siguiendo los parámetros de la Tabla 7.3.1 obtenemos unos resultados por el sistema EMVS con mucho ruido y poca precisión en la estimación, debido a la cantidad alta de eventos a estimar profundidad 7.83, y la estimación de profundidad en colores 7.84.

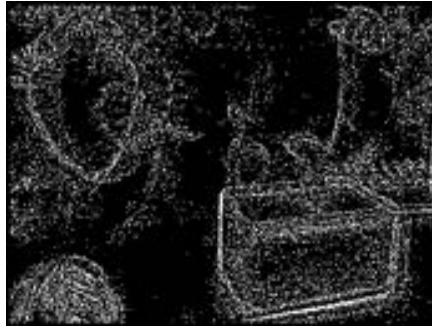


Figura 7.83: Eventos recogidos para la reconstrucción 3D de la prueba con objetos pequeños en escritorio



Figura 7.84: Mapa de profundidad a color de la prueba con objetos pequeños en escritorio

Parámetros de configuración para el EMVS	
<b>adaptive_threshold_kernel_size</b>	17
<b>adaptive_threshold_c</b>	15

Cuadro 7.8: Parámetros de configuración añadidos adicionalmente para la prueba con objetos pequeños en escritorio

Estos resultados se pueden intentar limpiar con el uso de unos parámetros de configuración adicionales para el reconstructor 3D. Para ello, se añaden los parámetros indicados en la Tabla 7.3.1, “adaptive\_threshold\_kernel\_size”, por defecto es una matriz kernel de 5x5 y la aumentamos de tamaño a una 17x17, y el parámetro “adaptive\_threshold\_c”, un valor entre 0 y 255, por defecto también 5, un menor valor en esta variable se traducirá en una

reconstrucción más densa y ruidosa. Por lo tanto, se aumenta la variable al valor de 15, intentando reducir así el ruido en la imagen de profundidad, además, con un tamaño del kernel mayor para una mayor concentración de los eventos en los bordes.

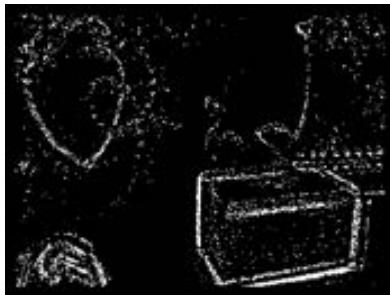


Figura 7.85: Eventos recogidos con parámetros extra para la reconstrucción 3D de la prueba con objetos pequeños en escritorio

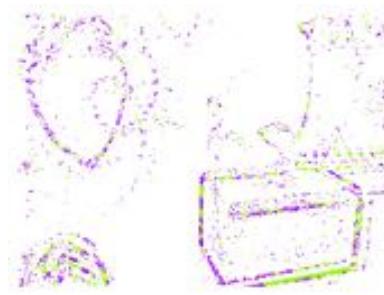


Figura 7.86: Mapa de profundidad a color extraído con parámetros extra de la prueba con objetos pequeños en escritorio

Con estos nuevos parámetros, los resultados pueden observarse de mejor manera los eventos [7.85](#), concentrados en los bordes de los objetos, y una mayor claridad para observar la profundidad estimada en los eventos [7.86](#). A pesar de ello, vemos como el corazón de la izquierda concentra la mayor cantidad de eventos en la capa más profunda, y ambos objetos que se encuentran enfrente más próximos a la cámara, pero tienen una combinación de eventos de la capa más cercana y la más alejada.

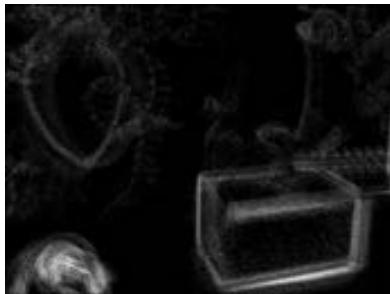


Figura 7.87: Mapa de confianza en la estimación de la profundidad de cada evento en la prueba con objetos pequeños en escritorio

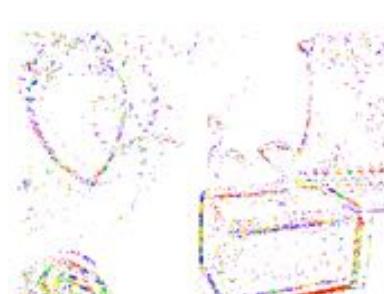


Figura 7.88: Mapa de profundidad a color con 200 capas de la prueba con objetos pequeños en escritorio

Para observar si permitiendo una mayor cantidad de capas es capaz de mostrar mejores estimaciones, se amplía la variable “dimZ” a 200 capas. En la Figura [7.88](#), los colores cálidos pertenecen a la zona más próxima a la

cámara (rojo-naranja-amarillo-verde) y conforme más fríos los colores, más alejados (azul-morado). Vemos cómo al aumentar la cantidad de capas para estimar la profundidad, se mantienen los resultados en los lugares donde se concentraban muchos eventos pertenecientes a una capa en la prueba anterior, como el borde inferior de la caja predomina el rojo, o el borde izquierdo más alejado de la caja tiene una gran cantidad de eventos azules. También la zona de los eventos pertenecientes al fondo suele destacar los eventos con el color morado, la capa más alejada de profundidad.

Parámetros de reducción de ruido para la nube de eventos	
radius_search	0.05
min_num_neigbors	1200

Cuadro 7.9: Parámetros de reducción de ruido para la nube de eventos para la prueba con objetos pequeños en escritorio

Además en esta prueba al tener una gran densidad de eventos en la reconstrucción 3D, se pueden añadir unos filtros que reduzcan el ruido en la nube de puntos. Estos parámetros Tabla 7.3.1, marcan el radio del área en el que se comprueban el mínimo de puntos vecinos especificados, en nuestro caso 1200 puntos en un radio de 0.05. Esto se traduce en una reducción de puntos aislados en la nube de 1137 puntos, como se muestra en el mensaje por terminal 7.89.

```
I0724 11:30:52.275386 1408133 mapper_emvs.cpp:316] Nube de puntos tiene antes de la
reducción de ruido = 4681
I0724 11:30:52.820189 1408133 main.cpp:154] Saved 3544 data points to pointcloud.pc
d
```

Figura 7.89: Salida terminal creación nube de puntos antes y después de la reducción de ruido

A pesar de ello, la nube de puntos es poco representativa de la escena real grabada, debido a la gran cantidad de eventos en la misma zona que se estiman en cada una de las distintas capas.

### Prueba cajas 1

En esta prueba se graban tres cajas con distinta altura situadas en el suelo, con imágenes de Aruco y grafitis que ayudan a la estimación de la cámara del sistema SVO 7.90. Esto provocará una alta generación de eventos, que se traducirá en una reconstrucción muy densa. Se usan los parámetros de la Tabla 7.3.1, dando una diferencia de profundidad de 15 cm entre el mínimo y máximo de profundidad en la escena y 3 capas de profundidad para estimar.

Parámetros de configuración para el EMVS	
<b>min_depth (metros)</b>	0.6
<b>max_depth (metros)</b>	0.75
<b>start_time_s (segundos)</b>	2.0
<b>stop_time_s (segundos)</b>	2.1
<b>dimZ (capas)</b>	3

Cuadro 7.10: Parámetros de configuración del EMVS para la prueba cajas 1

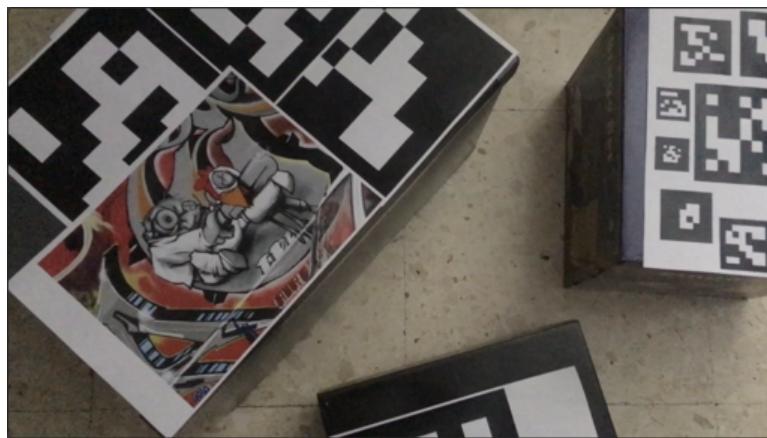


Figura 7.90: Fotograma correspondiente a la utilizada para la reconstrucción 3D de la prueba cajas 1

En el mapa de eventos [7.91] observamos como la zona del graffiti y la parte negra de los arucos tienen una gran cantidad de eventos, además de la sombra provocada por la caja situada más a la derecha de la imagen. Esto provoca una gran cantidad de eventos para evaluar una estimación de profundidad, como se ve en la Figura [7.92], donde los bordes de las imágenes de la caja izquierda, se estiman en la capa de profundidad más próxima a la cámara, y en el resto de cajas los eventos morados, pertenecientes a la capa más profunda.

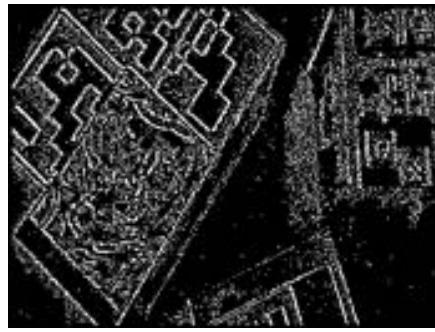


Figura 7.91: Eventos recogidos para la reconstrucción 3D de la prueba de cajas 1



Figura 7.92: Mapa de profundidad a color de la prueba de cajas 1

En la nube de puntos [7.93](#), donde a pesar de usar 3 capas de profundidad para la estimación, una de ellas con el color verde en la Figura [7.92](#), tras la reducción de ruido con unos 1000 puntos vecinos mínimos. Queda evidente que la caja de la izquierda se sitúa más cercana a la imagen que el resto, como así estima EMVS.

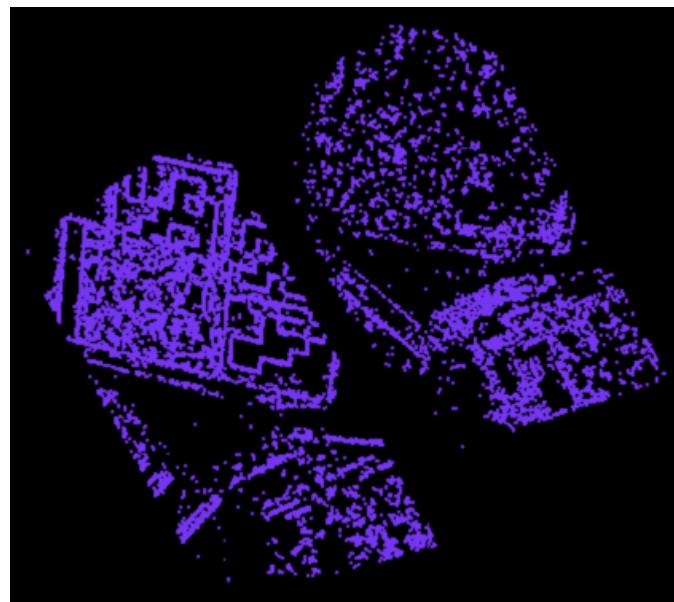


Figura 7.93: Nube de puntos de la prueba de cajas 1

### Prueba cajas 2

En esta prueba se mantiene la misma configuración que en la prueba anterior, pero con una disposición diferente en el intervalo tomado para la

reconstrucción 3D como se muestra en la Figura 7.94.



Figura 7.94: Fotograma en el intervalo seleccionado para la reconstrucción 3D de la prueba con cajas 2

Parámetros de configuración para el EMVS	
<b>min_depth (metros)</b>	0.6
<b>max_depth (metros)</b>	0.75
<b>start_time_s (segundos)</b>	14.0
<b>stop_time_s (segundos)</b>	14.5
<b>dimZ (capas)</b>	4

Cuadro 7.11: Parámetros de configuración del EMVS para la prueba cajas 2

En la Tabla 7.3.1 se muestran los parámetros usados para realizar la reconstrucción 3D, muy similares a los de la prueba anterior, con una capa más de profundidad y un intervalo de 5 décimas de segundo.



Figura 7.95: Eventos recogidos para la reconstrucción 3D de la prueba de cajas 2



Figura 7.96: Mapa de profundidad a color de la prueba de cajas 2

Como en la prueba anterior, la cantidad de eventos a estimar la profundidad es elevada [7.95], y da lugar a una representación muy ruidosa como se observa en la Figura [7.96]. Podemos ver como el sistema EMVS es capaz de estimar la profundidad correcta en la caja situada en la parte inferior de la imagen, ya que es la que menos altura tiene de las tres cajas. A pesar de ello, la cantidad de ruido en las otras dos cajas no permite una evaluación precisa, ya que existen eventos en cada una de las 4 capas de profundidad, pero podemos observar como no predomina el morado en ambas cajas, estimando que tienen una profundidad más cercana a la cámara que la caja inferior.



Figura 7.97: Eventos recogidos con menor ruido para la reconstrucción 3D de la prueba de cajas 2

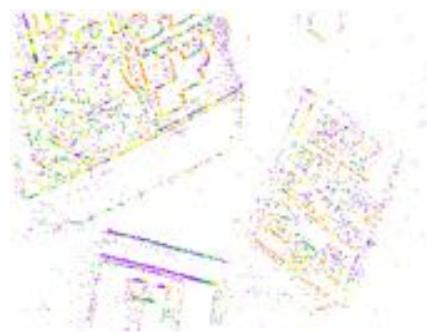


Figura 7.98: Mapa de profundidad a color con menor ruido de la prueba de cajas 2

Se intenta reducir la densidad de eventos en la imagen para una mejor compresión, para ello se sube el valor del parámetro de configuración “adaptive\_threshold\_c” de 5 a 9, dejando una reconstrucción menos densa y ruidosa [7.97], y se amplia a 10 capas de profundidad a estimar, dando un resultado más limpio, que se puede apreciar de mejor manera la estimación de la profundidad en las cajas [7.98], donde ya sí se puede observar como las cajas con mayor altura, reciben una estimación de las capas más cercanas a la cámara.

### 7.3.2. Ejemplos Dataset Zurich

#### Shapes

En esta prueba se graba un vídeo enfocando una pared con distintas imágenes de formas geométricas distintas [7.99], los parámetros de configuración se eligen los seleccionados por los propios autores Tabla [7.3.2], a los que se estiman la profundidad de los puntos de la Figura [7.100], dando como resultado el mapa de profundidad a color [7.101].



Figura 7.99: Entorno grabado para la prueba “Shapes” de la universidad de Zurich

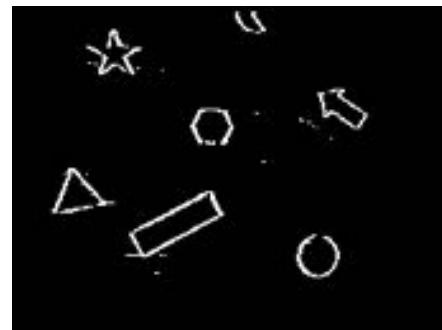


Figura 7.100: Eventos recogidos para la reconstrucción 3D para la prueba “Shapes” de la universidad de Zurich

Parámetros de configuración para el EMVS	
min_depth (metros)	0.5
max_depth (metros)	3.0
start_time_s (segundos)	2.0
stop_time_s (segundos)	4.0
dimZ (capas)	100

Cuadro 7.12: Parámetros de configuración del EMVS para la prueba “Shapes” de la universidad de Zurich

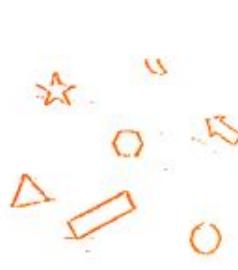


Figura 7.101: Mapa de profundidad a color para la prueba “Shapes” de la universidad de Zurich

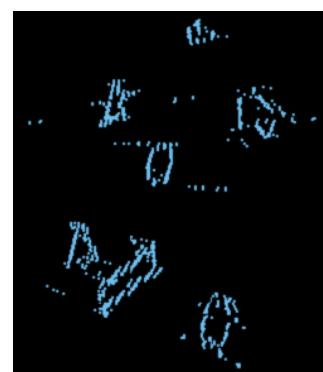


Figura 7.102: Nube de puntos para la prueba “Shapes” de la universidad de Zurich

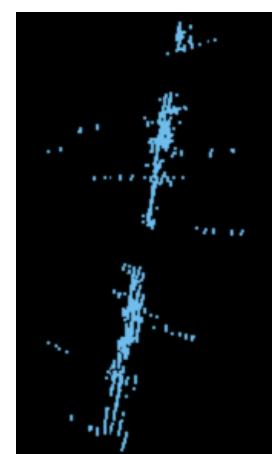


Figura 7.103: Nube de puntos para la prueba “Shapes” de la universidad de Zurich

La nube de puntos generada [\[7.102\]](#), muestra como a pesar de seleccionar 100 capas de profundidad distintas, el sistema EMVS es capaz de estimar la profundidad de las formas geométricas en las mismas capas de profundidad, como mejor se puede apreciar en la Figura [\[7.103\]](#), donde todas las figuras geométricas se encuentran en el mismo plano de profundidad, quedando fuera de él los pocos eventos pertenecientes al ruido.

### Boxes

En esta prueba se graba un entorno con una gran variedad de cajas con distinto tamaño en una habitación [\[7.104\]](#), con los parámetros de configuración de la Tabla [\[7.3.2\]](#).



Figura 7.104: Entorno grabado para la prueba “Boxes” de la universidad de Zurich

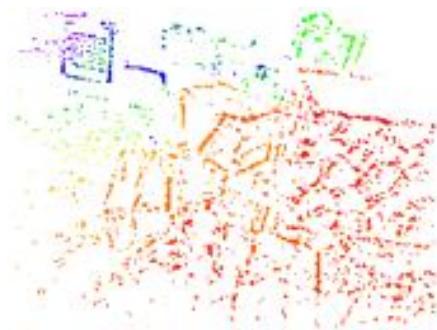


Figura 7.105: Mapa de profundidad a color para la prueba “Boxes” de la universidad de Zurich

Parámetros de configuración para el EMVS	
<b>min_depth (metros)</b>	1.0
<b>max_depth (metros)</b>	6.0
<b>start_time_s (segundos)</b>	3.5
<b>stop_time_s (segundos)</b>	5.5
<b>dimZ (capas)</b>	100
<b>adaptive_threshold_c</b>	7
<b>median_filter_size</b>	7

Cuadro 7.13: Parámetros de configuración del EMVS para la prueba “Boxes” de la universidad de Zurich

En la Figura [\[7.105\]](#) se puede ver la estimación de profundidad a los eventos, donde las cajas situadas en la parte inferior de la imagen se encuentran más cercanas a la cámara (representado con los colores más cálidos), y el “tablero de ajedrez”, usado para calibración de cámaras, situado en la pared

recibe las capas de profundidad más alejadas de la cámara, con tonos fríos de color azul y morado.

### Slider depth

En esta prueba se graba un entorno con varios objetos encima de una mesa y una diana de dardos al final de la misma como se puede observar en la Figura 7.106, recoge los eventos 7.107 para estimarles su profundidad, obtenidos con los parámetros de configuración de la Tabla 7.3.2.

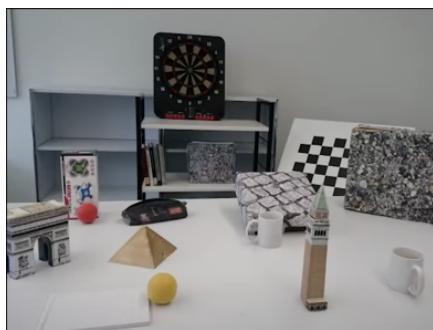


Figura 7.106: Entorno grabado para la prueba “Slider\_depth” de la universidad de Zurich



Figura 7.107: Eventos recogidos para la reconstrucción 3D para la prueba “Slider\_depth” de la universidad de Zurich

Parámetros de configuración para el EMVS	
<b>min_depth (metros)</b>	0.47
<b>max_depth (metros)</b>	2.4
<b>start_time_s (segundos)</b>	0.0
<b>stop_time_s (segundos)</b>	5.0
<b>dimZ (capas)</b>	100

Cuadro 7.14: Parámetros de configuración del EMVS para la prueba “Slider\_depth” de la universidad de Zurich

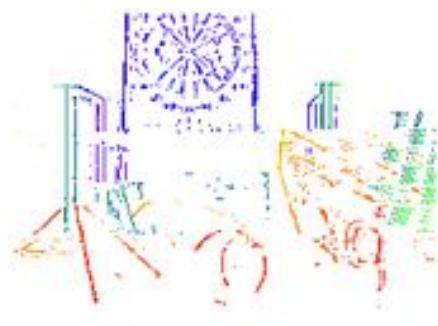


Figura 7.108: Mapa de profundidad a color para la prueba “Slider\_depth” de la universidad de Zurich

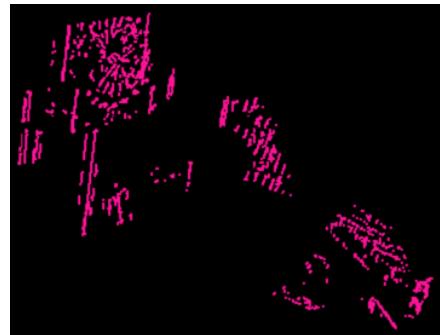


Figura 7.109: Nube de puntos para la prueba “Slider\_depth” de la universidad de Zurich con una vista lateral izquierda

La profundidad obtenida por el EMVS [7.108](#) da lugar a la nube de puntos de la Figura [7.109](#). Se puede observar como la zona de estantería donde se coloca encima la diana de dardos se encuentra al fondo, estimando con precisión la profundidad de la estantería sin incluir todos los puntos en la misma capa de profundidad, como sí ocurre en la diana. Además, sitúa los objetos en posiciones acordes al entorno real.

### Dynamic

En esta prueba se graba una oficina con una persona sentada en ella [7.110](#), que con los parámetros de la Tabla [7.3.2](#) obtiene los puntos que se observan en la Figura [7.111](#) para estimar la profundidad.

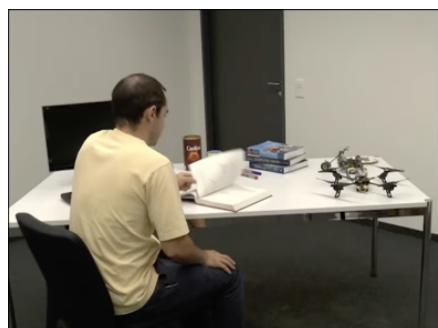


Figura 7.110: Entorno grabado para la prueba “Dynamic” de la universidad de Zurich



Figura 7.111: Eventos recogidos para la reconstrucción 3D para la prueba “Dynamic” de la universidad de Zurich

Parámetros de configuración para el EMVS	
min_depth (metros)	1.1
max_depth (metros)	2.2
start_time_s (segundos)	6.2
stop_time_s (segundos)	8.2
dimZ (capas)	100
adaptive_threshold_c	6
median_filter_size	15
radius_search	0.05
min_num_neighbors	3

Cuadro 7.15: Parámetros de configuración del EMVS para la prueba “Dynamic” de la universidad de Zurich

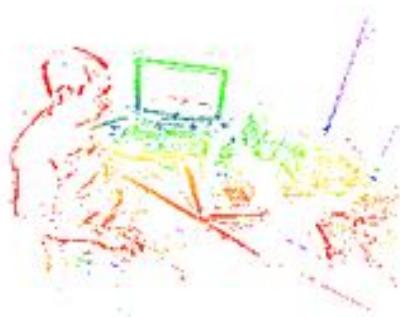


Figura 7.112: Mapa de profundidad a color para la prueba “Dynamic” de la universidad de Zurich

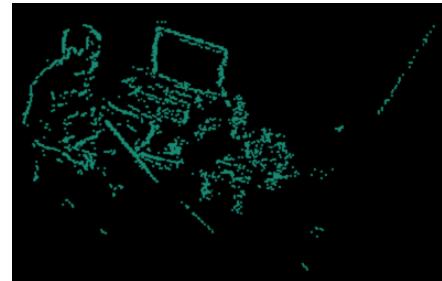


Figura 7.113: Nube de puntos para la prueba “Dynamic” de la universidad de Zurich

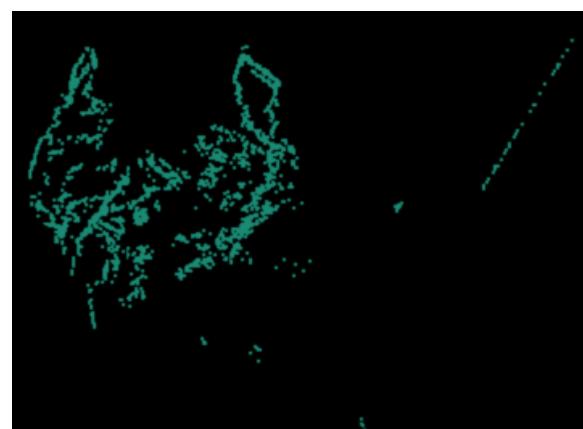


Figura 7.114: Nube de puntos para la prueba “Dynamic” de la universidad de Zurich con una toma lateral derecha

La imagen de estimación de profundidad [7.112](#), estima de forma adecuada las posiciones de la persona, objetos como el libro de la mesa, de los más cercanos a la cámara, los bordes de la puerta al fondo del entorno y el monitor como el objeto más alejado del escritorio. Esto se puede ver en la Figura [7.114](#), con una perspectiva que facilita la visualización lateral de la escena con respecto a la imagen original.

### 7.3.3. Discusión resultados Zurich - Propios

Una vez hechas estas pruebas para el reconstructor 3D EMVS con vídeos personales y el uso de los sistemas V2E, para la generación de eventos, y SVO, para la estimación de las posiciones de la cámara, comprendemos que el uso de vídeos de una alta resolución provoca una gran cantidad de eventos, que implica un mayor ruido e imprecisión en la estimación de profundidad. Además, los tiempos para evaluar el mapa de disparidad en la imagen o DSI en la resolución 346x260 píxeles de entre 2 a 15 milisegundos, mientras que los ejemplos a resolución 720p tardan entre 38 a 50 milisegundos. Siendo tres veces más rápido en la resolución más baja, y por tanto más eficaz en aplicaciones en tiempo real.

Los intervalos de tiempo escogidos con el conjunto de datos de la universidad de Zurich puede variar en todo el fichero, sin embargo, para las pruebas personales que se han realizado se necesita encontrar intervalos de tiempo de pequeña duración, entre 1 décima de segundo a 1 segundo de duración, y funciona solo en algunos momentos del vídeo. Esto sucede por los sistemas usados en el proyecto, V2E la simulación de eventos y SVO para la estimación de la posición de la cámara, ya que estas estimaciones pueden no ser lo precisas que se requieren, ya sea porque durante el seguimiento no llega al mínimo de características robustas a seguir o porque la estimación no sea lo suficientemente precisa para lo que necesita el EMVS.

Los datasets proporcionados por la universidad de Zurich obtienen la información a través de un sensor DVS, por lo tanto son los eventos reales y no un simulador, y otra diferencia clave es la obtención de las posiciones de la cámara, donde no la estiman con un algoritmo como SVO, sino que se apoyan de un dispositivo de captación de movimiento “OptiTrack” <sup>5</sup> para guardar las posiciones de la cámara.

Esas diferencias provocan que los datos de entrada proporcionados a EMVS con los vídeos personales que realizamos contienen una gran cantidad de ruido, sobre todo en la posición de la cámara, ya que es la razón por la cual los movimientos de los vídeos han de ser suaves, y al realizar desplazamien-

---

<sup>5</sup><https://optitrack.com/>

tos no es capaz de estimarlos correctamente el sistema SVO. Esto implica no poder realizar los movimientos necesarios para generar un DSi para el EMVS en mejores condiciones.

## **Capítulo 8**

# **Conclusiones y trabajo futuro**

En este capítulo se evalúa el trabajo realizado durante el Trabajo Fin de Grado, mostrando las conclusiones y trabajos futuros del proyecto. Durante las conclusiones se desarrolla una evaluación general de los objetivos conseguidos y los aspectos más relevantes del trabajo. Además, en este mismo capítulo se detallan los posibles avances futuros aplicables al proyecto.

### **8.1. Conclusiones**

El objetivo principal que perseguía el desarrollo de este proyecto era tratar la información que nos proporcionan los eventos para realizar una reconstrucción 3D del entorno.

Debido al carácter del trabajo, lo más adecuado era escoger un espacio de trabajo que facilitara el tratamiento de datos de los sensores DVS y la integración de distintas funcionalidades en un mismo sistema. Por ello se escogió el middleware ROS, que al no tener experiencia previa, comprender su funcionamiento ha resultado uno de los retos más importantes en el proyecto, ya que capacitó la comunicación entre los distintos sistemas y el uso de librerías propias a los sensores DVS.

Entre los distintos sistemas a incorporar al trabajo se encuentra el simulador de eventos V2E, que con la ausencia del sensor DVS para la realización del proyecto, se presentaba como una solución interesante por la versatilidad que nos ofrecía, pudiendo usar técnicas basadas en la información de los eventos, como también en métodos que hacen uso de las imágenes tradicionales.

La reconstrucción 3D la llevamos a cabo con el sistema EMVS, donde nos apoyamos en los datos simulados de los eventos, pero que además, requiere de las posiciones de la cámara para su correcto funcionamiento. Por la carencia de la posesión de un dispositivo que nos ayude a la obtención de esta información, hacemos uso del sistema SVO, que estima estas posiciones a partir de los fotogramas clásicos.

Al combinar estos sistemas haciendo uso de los mismos datos de entrada, surgieron muchos problemas a la hora de encontrar unos vídeos de entrada válidos para ambos sistemas, ya que las restricciones de cada uno de los sistemas son diferentes, necesitando distintos tipos de movimientos para una mayor eficiencia en los resultados, esto nos llevó a la búsqueda de unos vídeos capaces de encontrar un equilibrio en el correcto funcionamiento de ambos sistemas.

En adición, previamente al desarrollo de este Trabajo Fin de Grado, descubrí la existencia de estos sensores DVS y sus ventajas para los campos de visión por computador. Gracias a este proyecto, he podido profundizar en las aplicaciones y ventajas que proporcionan estos sensores con respecto a las cámaras tradicionales, ampliando mi conocimiento en el ámbito de la visión por computador con nuevas técnicas aplicables a eventos.

El uso de eventos en contraposición de los fotogramas clásicos, muestra un gran potencial para el desarrollo y resolución de varios paradigmas del campo de visión por computador, por la reducción de información redundante, bajando en algunos casos el tamaño del fotograma entre un 95 % y 99 % usando eventos frente a píxeles a color o escala de grises. Además de la asincronía con la que se captan los eventos, permitiendo detectar con mayor rapidez los cambios sufridos en el entorno debido a su alta resolución temporal.

He de concluir, por tanto, que una vez terminado el proyecto, tengo una sensación satisfactoria con los resultados obtenidos. El sistema integrado es capaz de estimar la profundidad en entornos, consiguiendo una reconstrucción 3D a través de los eventos con rendimiento en tiempo real, necesitando entre 3 a 60 milisegundos para su procesamiento. Así pues, se ha conseguido cumplir con los objetivos estipulados con el uso de los medios disponibles, contribuyendo al campo de desarrollo y dando pie a la ampliación y mejora del sistema.

## 8.2. Análisis de objetivos

Estudiar el estado del arte de los eventos y los métodos para su simulación, técnicas para la reconstrucción 3D y para la estimación de la posición de la cámara. Para ello se ha consultado en distintos documentos científicos, como distintos artículos publicados, libros o estudios, de manera que se han alcanzado los conocimientos necesarios sobre los sensores de eventos y las distintas aplicaciones existentes para generar las funcionalidades requeridas en el proyecto.

- **Construir un sistema capaz de realizar una reconstrucción de un entorno en tres dimensiones a partir de datos de un sensor de eventos.** Para cumplir este objetivo se han estudiado las distintas técnicas disponibles para la reconstrucción 3D a través de eventos. Escogiendo los que mejor se ajustan al proyecto y a los medios disponibles.
- **Estudiar las ventajas de usar un simulador de eventos.** Se estudiaron los distintos simuladores de eventos disponibles y se eligió el V2E, además, se realizaron distintas comparaciones entre los fotogramas de eventos y los fotogramas tradicionales con la realización de distintas gráficas y tablas con los datos obtenidos.
- **Estudiar las funcionalidades y herramientas que ofrece el middleware ROS.** Para cumplir con este objetivo se ha hecho un estudio integral de esta plataforma a través de foros, ejemplos en internet y la propia documentación oficial con distintos tutoriales para la comprensión de las distintas funcionalidades y componentes de ROS.
- **Desarrollar un paquete software para integrar los diferentes módulos del sistema.** Se ha conseguido desarrollar un módulo capaz de recibir la información de los sistemas V2E y SVO, para posteriormente transmitirla al reconstructor 3D EMVS con el formato necesario para la correcta comprensión de la información por el sistema.
- **Realizar un sistema en código abierto utilizando herramientas de software libre disponible para cualquier usuario que lo requiera.** El sistema está disponible en una plataforma GitHub [41] con licencia GPL-3.0 License, junto a un manual de usuario que simplifica su instalación y uso.
- **Desarrollar un sistema extensible, capaz de adoptar de forma sencilla nuevas funcionalidades al sistema.** La modularidad del sistema permite la extensión del proyecto de forma sencilla y rápida sin la necesidad de realizar cambios al proyecto original.

### 8.3. Trabajo futuro

Este proyecto ofrece un gran número de posibilidades en lo que respecta a posibles ampliaciones y nuevas funcionalidades, pues existe una gran variedad de aplicaciones y usos para una reconstrucción 3D de un entorno.

Actualmente el trabajo está diseñado para poder incluir cualquier vídeo grabado con una cámara tradicional, simular los eventos y estimar la posición de la cámara a través de los fotogramas, para después realizar la reconstrucción 3D con la información resultante.

La principal mejora para el uso del sistema se propone usar un sensor DVS en lugar de simular los eventos, además de un dispositivo capaz de otorgar la posición de la cámara mientras se graba la escena, como lo es OptiTrack usado por el departamento de informática de la universidad de Zurich en su conjunto de datos. De esta manera se aseguran unas estimaciones de las posiciones de la cámara más precisas ayudando a una reconstrucción 3D más fiel.

Asimismo, se pueden integrar en el proyecto nuevas funcionalidades que amplíen el campo de trabajo en el que se desenvuelve el sistema, dado que se ha desarrollado de manera modular, no es difícil ampliar con nuevos módulos el proyecto sin afectar al funcionamiento del resto del sistema.

En general, se podrían desarrollar las siguientes ideas que se detallan a continuación:

- Desarrollo de una red neuronal capaz de detectar o reconocer en la nube de puntos objetos clave o personas, para operaciones de rescate en entornos dinámicos con el uso de drones equipados con un sensor DVS.
- Dotar a un sistema de navegación autónoma de inteligencia para que muestre un camino o sorteé obstáculos en el entorno.

Además debemos estar atentos a los avances en el estado del arte para la mejora del proyecto, como el uso de un estimador de la posición del sensor a partir de eventos y así trabajar exclusivamente con la información captada por un sensor DVS.

# Bibliografía

- [1] B. Starly, Z. Fang, W. Sun, A. Shokoufandeh, and W. Regli, “Three-dimensional reconstruction for medical-cad modeling,” *Computer-Aided Design and Applications*, vol. 2, pp. 431–438, 08 2013.
- [2] M.-D. Yang, C.-F. Chao, K.-S. Huang, L.-Y. Lu, and Y.-P. Chen, “Image-based 3d scene reconstruction and exploration in augmented reality,” *Automation in Construction*, vol. 33, p. 48–60, 08 2013.
- [3] A. Kargas, G. Loumos, and D. Varoutas, “Using different ways of 3d reconstruction of historical cities for gaming purposes: The case study of nafplio,” *Heritage*, vol. 2, pp. 1799–1811, 07 2019.
- [4] C. Poullis and S. You, “3d reconstruction of urban areas,” 05 2011, pp. 33–40.
- [5] M. Hejrati and D. Ramanan, “Analysis by synthesis: 3d object recognition by object reconstruction,” 06 2014, pp. 2449–2456.
- [6] “Filtro de kalman.” [Online]. Available: <https://www.kalmanfilter.net/default.aspx>
- [7] D. Chekhlov, M. Pupilli, W. Mayol-Cuevas, and A. Calway, “Real-time and robust monocular slam using predictive multi-resolution descriptors,” 11 2006, pp. 276–285.
- [8] R. Martinez-Cantin and J. Castellanos, “Unscented slam for large-scale outdoor environments,” 09 2005, pp. 3427 – 3432.
- [9] “Slam visual.” [Online]. Available: [https://es.wikipedia.org/wiki/SLAM\\_visual](https://es.wikipedia.org/wiki/SLAM_visual)
- [10] J. Engel, T. Schoeps, and D. Cremers, “Lsd-slam: large-scale direct monocular slam,” vol. 8690, 09 2014, pp. 1–16.
- [11] “Ros.” [Online]. Available: <https://www.ros.org>

- [12] H. Pham, H. La, D. Feil-Seifer, and L. Nguyen, “Reinforcement learning for autonomous uav navigation using function approximation,” 08 2018, pp. 1–6.
- [13] A. Nemati, M. Sarim, M. Hashemi, E. Schnipke, S. Reidling, W. Jeffers, J. Meiring, P. Sampathkumar, and M. Kumar, “Autonomous navigation of uav through gps-denied indoor environment with obstacles,” 01 2015.
- [14] P. Alliez, F. Bonardi, S. Bouchafa, J. Didier, H. Hadj-Abdelkader, F. I. I. Muñoz, V. Kachurka, B. Rault, M. Robin, and D. Roussel, “Real-time multi-slam system for agent localization and 3d mapping in dynamic scenarios,” *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4894–4900, 2020.
- [15] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, “Event-based vision: A survey,” 01 2019.
- [16] P. Lichtsteiner, C. Posch, and T. Delbruck, “A  $128 \times 128$  120 db 15 latency asynchronous temporal contrast vision sensor,” *Solid-State Circuits, IEEE Journal of*, vol. 43, pp. 566 – 576, 03 2008.
- [17] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, “The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam,” *The International Journal of Robotics Research*, vol. 36, 11 2016.
- [18] M. Mahowald, “Vlsi analogs of neuronal visual processing: a synthesis of form and function,” 1992.
- [19] C. Posch, D. Matolin, and R. Wohlgenannt, “A qvga 143db dynamic range asynchronous address-event pwm dynamic image sensor with lossless pixel-level video compression,” vol. 1, 03 2010, pp. 400 – 401.
- [20] C. Brändli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, “A  $240 \times 180$  130 db 3  $\mu$ s latency global shutter spatiotemporal vision sensor,” *Solid-State Circuits, IEEE Journal of*, vol. 49, pp. 2333–2341, 10 2014.
- [21] “Samsung smartthings vision.” [Online]. Available: <https://www.samsung.com/au/smart-home/smartthings-vision-u999/GP-U999GTEEAAC>
- [22] H. Rebecq, D. Gehrig, and D. Scaramuzza, “Esim: an open event camera simulator,” 10 2018.
- [23] D. Gehrig, M. Gehrig, J. Hidalgo-Carrio, and D. Scaramuzza, “Video to events: Recycling video datasets for event cameras,” 06 2020, pp. 3583–3592.

- [24] “Bt.709.” [Online]. Available: <https://www.itu.int/rec/R-REC-BT.709>
- [25] H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz, “Super slomo: High quality estimation of multiple intermediate frames for video interpolation,” 06 2018, pp. 9000–9008.
- [26] M. Zollhöfer, “Commodity rgb-d sensors: Data acquisition,” 02 2019.
- [27] M. Cook, L. Gugelmann, F. Jug, C. Krautz, and A. Steger, “Interacting maps for fast visual interpretation,” *Proceedings of the International Joint Conference on Neural Networks*, 07 2011.
- [28] D. Weikersdorfer, R. Hoffmann, and J. Conradt, “Simultaneous localization and mapping for event-based vision systems,” vol. 7963, 09 2013, pp. 133–142.
- [29] D. Weikersdorfer, D. B. Adrian, D. Cremers, and J. Conradt, “Event-based 3d slam with a depth-augmented dynamic vision sensor,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 359–364.
- [30] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza, “Low-latency visual odometry using event-based feature tracks,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 16–23.
- [31] H. Kim, S. Leutenegger, and A. Davison, “Real-time 3d reconstruction and 6-dof tracking with an event camera,” vol. 9910, 10 2016, pp. 349–364.
- [32] H. Rebecq, T. Horstschaefner, G. Gallego, and D. Scaramuzza, “Evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real-time,” *IEEE Robotics and Automation Letters*, vol. PP, 12 2016.
- [33] S. Baker, R. Gross, and I. Matthews, “Lucas-kanade 20 years on: A unifying framework: Part 3,” *Int. J. Comput. Vis.*, vol. 56, 12 2003.
- [34] G. Klein and D. Murray, “Parallel tracking and mapping for small ar workspaces,” 12 2007, pp. 225–234.
- [35] C. Forster, M. Pizzoli, and D. Scaramuzza, “Svo: Fast semi-direct monocular visual odometry,” 05 2014.
- [36] J. Kogler, C. Sulzbachner, M. Humenberger, and F. Eibenstein, *Address-Event Based Stereo Vision with Bio-Inspired Silicon Retina Imagers*, 01 2011.

- [37] P. Rogister, R. Benosman, S.-H. Ieng, P. Lichtsteiner, and T. Delbruck, “Asynchronous event-based binocular stereo matching,” *IEEE transactions on neural networks and learning systems*, vol. 23, pp. 347–353, 02 2012.
- [38] H. Rebecq, G. Gallego, E. Mueggler, and D. Scaramuzza, “Emvs: Event-based multi-view stereo—3d reconstruction with an event camera in real-time,” *International Journal of Computer Vision*, vol. 126, 12 2018.
- [39] S. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, “A comparison and evaluation of multi-view stereo reconstruction algorithms,” in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*, vol. 1, 2006, pp. 519–528.
- [40] “Voxel.” [Online]. Available: <https://es.wikipedia.org/wiki/Voxel>
- [41] “Repositorio github tfg.” [Online]. Available: [https://github.com/Joseadr99/TFG-3D\\_reconstruction/](https://github.com/Joseadr99/TFG-3D_reconstruction/)
- [42] R. S. Pressman, *Ingeniería del software;: un enfoque práctico*. McGraw Hill/Interamericana de España, 2002.
- [43] “Cuda.” [Online]. Available: <https://developer.nvidia.com/cuda-zone>
- [44] “Sueldo ingeniero informático.” [Online]. Available: <https://universidadeuropea.com/blog/cuanto\protect\discretionary{\char\nobreak\hyphenchar\font}{}{}gana\protect\discretionary{\char\nobreak\hyphenchar\font}{}{}un\protect\discretionary{\char\nobreak\hyphenchar\font}{}{}ingeniero\protect\discretionary{\char\nobreak\hyphenchar\font}{}{}informatico>
- [45] “Cameralinfo.” [Online]. Available: [http://docs.ros.org/en/melodic/api/sensor\\_msgs/html/msg/CameraInfo.html](http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/CameraInfo.html)
- [46] “Event.” [Online]. Available: [https://github.com/uzh\protect\discretionary{\char\nobreak\hyphenchar\font}{}{}rpg/rpg\\_dvs\\_ros/blob/master/dvs\\_msgs/msg/Event.msg](https://github.com/uzh\protect\discretionary{\char\nobreak\hyphenchar\font}{}{}rpg/rpg_dvs_ros/blob/master/dvs_msgs/msg/Event.msg)
- [47] “Eventarray.” [Online]. Available: [https://github.com/uzh\protect\discretionary{\char\nobreak\hyphenchar\font}{}{}rpg/rpg\\_dvs\\_ros/blob/master/dvs\\_msgs/msg/EventArray.msg](https://github.com/uzh\protect\discretionary{\char\nobreak\hyphenchar\font}{}{}rpg/rpg_dvs_ros/blob/master/dvs_msgs/msg/EventArray.msg)
- [48] “PoseStamped.” [Online]. Available: [http://docs.ros.org/en/noetic/api/geometry\\_msgs/html/msg/PoseStamped.html](http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/PoseStamped.html)
- [49] “Catkin.” [Online]. Available: [http://wiki.ros.org/catkin/conceptual\\_overview](http://wiki.ros.org/catkin/conceptual_overview)
- [50] “Svo.” [Online]. Available: [https://github.com/uzh-rpg/rpg\\_svo](https://github.com/uzh-rpg/rpg_svo)

- [51] “Video2bag.” [Online]. Available: [https://stackoverflow.com/questions/31432870/  
how-do-i-convert-a-video-or-a-sequence-of-images-to-a-bag-file](https://stackoverflow.com/questions/31432870/how-do-i-convert-a-video-or-a-sequence-of-images-to-a-bag-file)
- [52] “Emvs.” [Online]. Available: [https://github.com/uzh-rpg/rpg\\_emvs](https://github.com/uzh-rpg/rpg_emvs)
- [53] “Natural point.” [Online]. Available: [www.naturalpoint.com](http://www.naturalpoint.com)
- [54] “V2e.” [Online]. Available: <https://github.com/SensorsINI/v2e>
- [55] “Esim.” [Online]. Available: [https://github.com/uzh-rpg/rpg\\_esim](https://github.com/uzh-rpg/rpg_esim)
- [56] “Vikit.” [Online]. Available: [https://github.com/uzh-rpg/rpg\\_vikit](https://github.com/uzh-rpg/rpg_vikit)

