Universidad de Alcalá

Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de la Telecomunicación

Trabajo Fin de Grado

Seguimiento de personas para detección de caídas a partir de información de profundidad

Autor: Christian Reyes Sáez

Tutor/es: Cristina Losada Gutiérrez

2020

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de la Telecomunicación

Trabajo Fin de Grado

“Seguimiento de personas para detección de caídas a partir de información de profundidad”

Autor: Christian Reyes Sáez

Tutora: Cristina Losada Gutiérrez

TRIBUNAL:
Presidente: Javier Macías Guarasa
Vocal 1º: Ignacio Fernandez Lorenzo
Vocal 2º: Cristina Losada Gutiérrez

FECHA:

Agradecimientos

Aunque ya no tengamos aquella fuerza que antaño removía
cielo y tierra, seguimos siendo lo que somos:
el mismo temple en nuestros heroicos corazones
debilitados por el tiempo y el destino, pero con la firme voluntad de pelear,
de buscar, de encontrar, y de nunca rendirse.

Lord Tennyson – Poema *Ulysses*

Tras estos años inolvidables de mi vida, solo tengo que decir gracias a todos con los que he compartido mi viaje. Aún recuerdo aquella primera clase de universidad, física, equivocándonos de aula y de materia obviamente, con Aida. Los siguientes años te convertiste en mi amiga y confidente. A ti Gene, como mi compañera contra las noches en vela trabajando y de locura delante de la pantalla con el cursor parpadeando.

A Fer, como nos conocimos en el fragor de la desesperación pre-examen. Te has convertido en un hermano, en un pilar que me ha acompañado en mis horas más bajas y solo puedo decirte que tienes mi entera gratitud.

A ti Ly, me has acompañado en distintas partes de este largo camino, de distintas formas, y ahora teniéndote a mi lado me siento imparable.

Y a mi familia, me habéis aguantado, soportado el mal humor, me habéis visto con ojeras, las Navidades perdidas y solo puedo decir que siempre habéis sido un motor para mí. Y no me olvido de ti, aunque ya no estés aquí, siempre serás *mi sol en la tormenta* y atesoro en mi corazón tu mirada al verme en la foto de la Orla.

Resumen

El objetivo de este trabajo es el diseño, implementación y validación de un algoritmo para el seguimiento y detección de acciones de personas, partir de imágenes de profundidad obtenidas con un sensor basado en tiempo de vuelo (ToF: Time of Flight) comercial ubicado en posición cenital. Para ello se ha implementado una solución que detecta las personas presentes en la escena, realiza un seguimiento de su posición e infiere la acción que se está realizando. Toda la solución se encuentra desplegada dentro de un contenedor de software (Docker).

La solución se ha evaluado con videos de distintas personas, obteniendo en la detección de personas una sensibilidad del 97% y en la detección de acciones entorno al 89%.

Palabras clave: Cámaras de tiempo de vuelo, detección de personas, detección de acciones, Docker

Abstract

The objective of this work is the design, implementation, and validation of an algorithm for the monitoring and detection of human actions, based on depth images obtained with a commercial Time of Flight (ToF) camera located in top-view configuration. For this purpose, it has been designed and implemented a solution that detects the people present in the scene, tracks their position, and infers the action that is being carried out. The entire solution is deployed within a container solution (Docker).

The solution has been evaluated with videos of different people, obtaining a sensitivity of 97% in the people detection task, and around an 89% in the action recognition.

Keywords: Time of Flight, detection of humans, detection of human's actions, Docker

Índice

Índice de figuras

Índice de tablas

Capítulo 1.

Introducción.

Este trabajo fin de grado (TFG) se enmarca en las líneas de investigación del grupo GEINTRA [1] (Grupo de Ingeniería Electrónica aplicada a Espacios Inteligentes y Transporte) y toma como punto de partida los trabajos previos realizados en el grupo cuyo objetivo era la detección de personas a partir de información de profundidad [2] [3].

Partiendo de que se conoce la posición que ocupan las personas en la escena, el objetivo de este trabajo es incorporar un sistema de seguimiento robusto que permita tanto determinar la trayectoria seguida por las mismas en 3 dimensiones, como detectar posibles caídas, todo ello empleando únicamente información de profundidad adquirida por un sensor basado en la tecnología de tiempo de vuelo [4] (ToF). Se trata de un tema de gran interés debido a las posibles aplicaciones prácticas en sistemas de videovigilancia, así como para la prevención de situaciones potencialmente peligrosas para las personas, por ejemplo, la detección de caídas de personas mayores que viven solas, o de los pacientes de un hospital.

Además, el uso de información de profundidad no permite reconocer la identidad de las personas, por tanto, el sistema puede funcionar preservando la privacidad de los usuarios. La cámara se encuentra en posición cenital, con el fin de evitar problemas de oclusión, además de favorecer su instalación, también protege la privacidad de las personas detectadas, al no ser posible conocer su identidad.

1.1. Objetivos

Como se ha comentado previamente, el objetivo de este trabajo es el diseño, implementación y validación de un algoritmo para el seguimiento en tres dimensiones de múltiples objetivos (personas), detectados previamente, para conocer su trayectoria y reconocer las acciones que están realizando (andar, correr, estar parado o caerse). Con el fin de llegar a dicho objetivo ha sido necesario cumplir los siguientes objetivos específicos:

- Estudio y puesta en marcha del detector de personas
- Estudio e implementación de un sistema de seguimiento de personas, previamente detectadas, basado en el filtro de Kalman.
- Análisis e implementación de la algoritmia necesaria para el seguimiento de un número variable de personas empleando un banco de filtros de Kalman.
- Incorporación de un módulo para el reconocimiento de acciones y la detección de caídas.
- Estudio y visualización de los resultados.
- Evaluación experimental exhaustiva y obtención de resultados.

1.2. Solución propuesta

La solución desarrollada consiste en cuatro módulos que se ejecutan de forma secuencial, y pueden observarse en el diagrama general de la Figura 1.1 las tareas que se llevan a cabo por cada imagen de entrada.

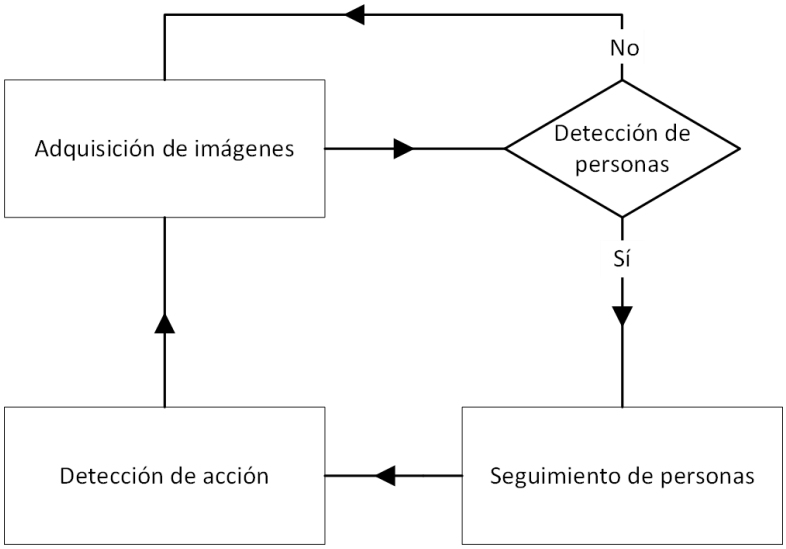

Figura 1.1 Diagrama general del proyecto.

En este trabajo, se ha partido de un conjunto de secuencias previamente grabadas y etiquetadas en las que diferentes personas realizan las distintas acciones a detectar. Por este motivo, en lugar de realizar la captura de imágenes en tiempo real, se ha implementado una primera etapa de lectura de la información de un archivo, en el que la distancia de cada punto a la cámara se representa en milímetros, como una variable de 16 bits sin signo. Mientras que la lectura de la información proveniente de la cámara sea válida se mantendrá la ejecución, es decir, el programa finalizara cuando el vídeo grabado lo haga. Cabe destacar que, debido a que el desarrollo se ha llevado a cabo de forma modular, es sencillo sustituir el módulo de lectura de archivo por una función que realice la captura en tiempo real de la cámara.

La información obtenida se analiza para evaluar si hay alguna persona, y se obtiene su posición 3D. En caso contrario, se ejecuta la siguiente interacción del programa.

El siguiente paso es el seguimiento de las personas en la escena mediante un banco de filtros de Kalman. En esta etapa se obtiene una estimación de la posición de las personas presentes en la escena (a partir de la información disponible de escenas anteriores) y se relaciona la información de dicha estimación con la posición de las personas detectadas, para la posterior corrección.

Con la información de seguimiento obtenida, se realiza un estudio y categorización de la velocidad, con el objetivo de determinar las distintas acciones que se pueden estar realizando: andar, andar despacio, correr, pararse y caerse. Toda la información se almacena en ficheros de texto para su posterior evaluación y representación.

1.3. Estructura del documento

El documento contiene cinco capítulos generales, en los que se referencia la introducción del desarrollo del proyecto (Capítulo 1) ,el estudio teórico de los fundamentos físicos y matemáticos sobre los que se apoya el trabajo desarrollado (Capítulo 2), las soluciones implementadas (Capítulo 3), los resultados obtenidos con las mismas (Capítulo 4) y por ultimo las conclusiones y líneas futuras para proseguir con el trabajo (Capítulo 5).

Capítulo 2.

Estudio teórico

Como ya se ha comentado en la Introducción, en el diagrama general de la Figura 2.1 se observa el objetivo de este trabajo, la identificación y seguimiento de personas, el reconocimiento de acciones y la detección de caídas, todo ello a partir de la información de profundidad proporcionada por un sensor ToF ubicado en posición cenital.

En este capítulo, se presenta el estudio teórico realizado para alcanzar la solución propuesta. En primer lugar, se describe el principio de funcionamiento de la cámara de profundidad empleada para la adquisición de imágenes, de forma específica se analiza la cámara comercial Kinect II de Microsoft [4]. A continuación, se detallan los principios matemáticos que se utilizan para realizar la predicción de la posición del objetivo, el filtro de Kalman y distintas técnicas en lo concerniente al reconocimiento de acciones. Posteriormente se presentan los principios SOLID [5] como patrón de diseño de software, como buenas prácticas a seguir a la hora de desarrollar software, y por último la solución de software libre usado para la virtualización de la propuesta.

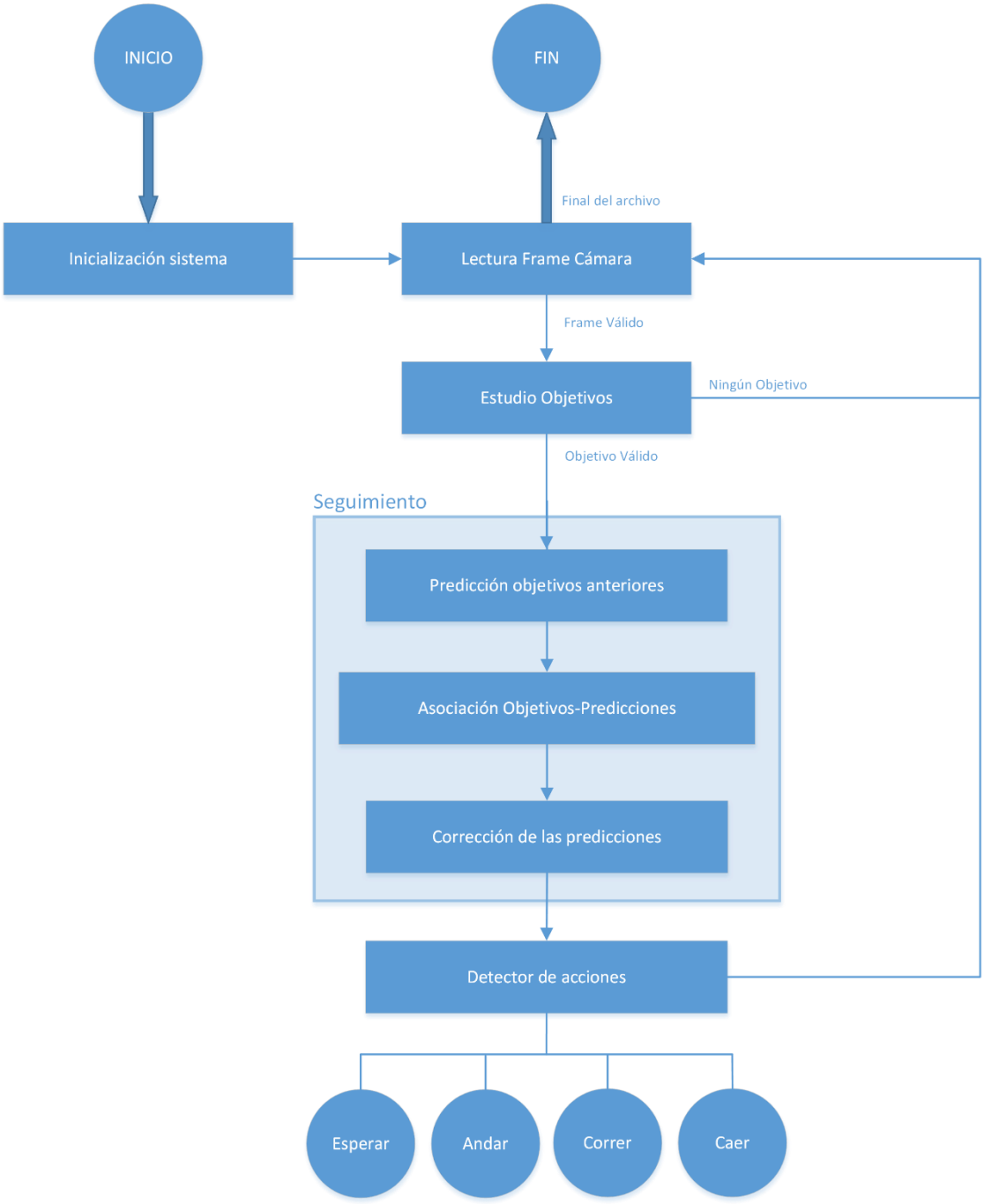

Figura 2.1 Diagrama de bloques de la solución propuesta.

2.1. Principios de funcionamiento de cámaras de tiempo de vuelo

La aparición de la tecnología ToF (ToF: Time of Flight) ha permitido estimar la distancia entre objeto y cámara a partir de imágenes de profundidad de la escena [6]. Este tipo de sensores comienza a emplearse como elemento indispensable en sistemas hombre-máquina, en la industria del entretenimiento como parte de sistemas de sensorizado en aplicaciones de realidad aumentada [7], o como ya se ha comentado, como parte del sistema de percepción de sistemas robóticos [8]. A veces se emplea como parte única del sistema sensorial y otras muchas en combinación con otros sensores. En aquellos casos, en los que la cámara ToF forma parte del sistema de percepción de un robot que se mueve, la escena varía y la distancia de los objetos presentes cambia continuamente.

La principal ventaja de una cámara ToF es la extracción de profundidad para determinar características geométricas y planos de la escena de trabajo. Además, una correcta calibración de ésta permite transformar la imagen de profundidades en un conjunto de puntos 3D del entorno, proporcionando así información espacial tridimensional. Por otro lado, el principal inconveniente son las fluctuaciones en la precisión con la que se mide la distancia al objetivo, principalmente debido a las interferencias causadas por factores externos, como por ejemplo la luz solar, la orientación, la reflectividad de la superficie de los objetos de la escena y la distancia de trabajo entre cámara y objetivo. En la literatura, son varios los autores que han hecho estudios que modelan y estiman el error en cámaras ToF [6].

Actualmente, los sensores ToF han alcanzado un gran impacto en diferentes áreas de investigación de la visión por computador y la percepción, desde el reconocimiento de objetos, la prevención de colisiones y/o la reconstrucción de escenas y/u objetos. En todos ellos se requiere medir distancias con una buena precisión. De ahí, que sean variados y numerosos los estudios cuyo objetivo es la obtención de métodos de calibración que permitan medir distancias a objetos con la mayor exactitud posible [9].

Estos experimentos y aplicaciones han dado lugar a sensores ToF de uso industrial (PMD, CSEM, etc.) así como de bajo coste y uso doméstico como Kinect de Microsoft [6].

Las cámaras ToF emplean luz modulada en amplitud. Estas cámaras montan cerca de la óptica, matrices de diodos LED que emiten señales con longitud de onda cercana al infrarrojo (Figura 2.2). Estas fuentes de luz son de tipo NIR (NIR: *Near Infrared*; infrarrojo cercano). Las cámaras de tiempo de vuelo basadas en modulación continua miden el tiempo que necesitan los haces de luz infrarroja en propagarse hasta la superficie del objeto y en regresar en forma de luz reflejada. En definitiva, estima el desplazamiento en fase mediante la correlación entre la señal reflejada y la señal empleada como referencia. A este fenómeno se le conoce como Interferometría de desplazamiento en fase (PSI: *Phase Shifting Interferometry*). Un estudio de interferometría en cámaras ToF es expuesto en la siguiente publicación [10].

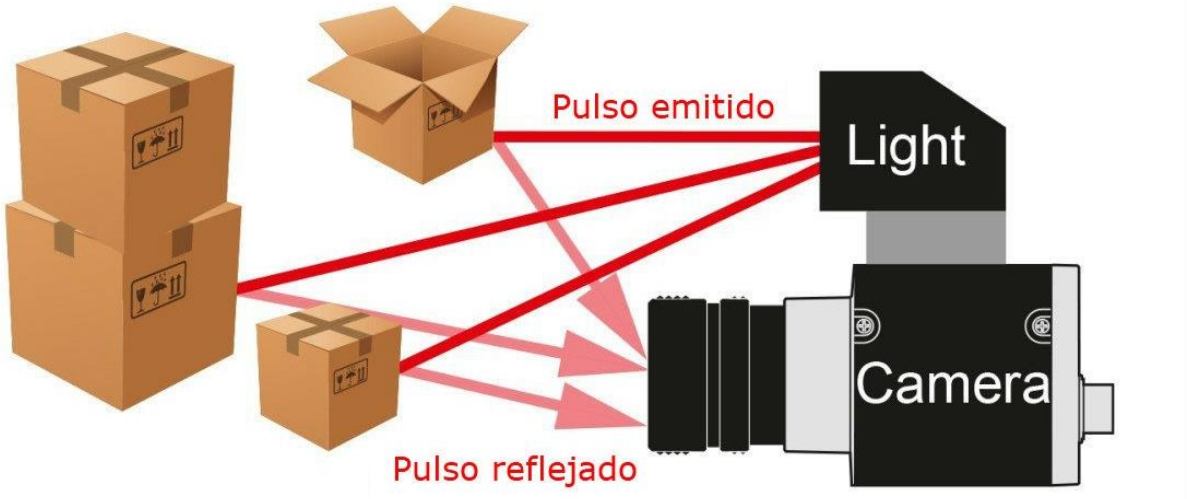

Figura 2.2 Esquema funcionamiento cámara NIR.

La correlación entre cuatro señales desplazadas fases de 90º, tal como se observa en la Figura 2.3 denotadas por r0 (0º), r1 (90º), r2 (180º), r3 (270º) y tomadas con un periodo de $T=1/f_{mod}$ permite calcular el retardo en fase, ϕ_i (ecuación 2.1) y la amplitud de la señal infrarroja recibida a_i (ecuación 2.2). A partir del desfase ϕ_i, es posible obtener de forma indirecta, el tiempo de vuelo de la señal infrarroja, y conocida la velocidad de la luz, *c* y la frecuencia de modulación, f_{mod}, puede calcularse la distancia d_i, en cada píxel *i* de los n píxeles que forman la imagen bidimensional de la siguiente manera mediante la ecuación 2.3.

$$\phi_i = \arctan\left(\frac{r_1 - r_3}{r_0 - r_2}\right) \quad 2.1$$

$$a_i = \frac{\sqrt{(r_1 - r_3)^2 + (r_0 - r_2)^2}}{2} \quad 2.2$$

$$d_i = \frac{c * \phi_i}{4\pi * f_{mod}} \quad 2.3$$

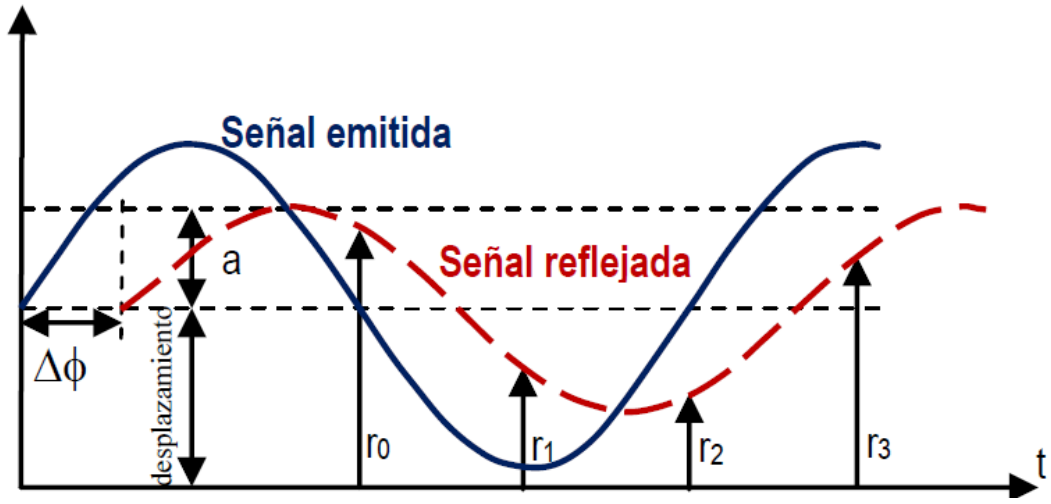

Figura 2.3 Señal emitida y reflejada por la cámara ToF.

Las cámaras de tiempo de vuelo proporcionan un mapa de distancias (d_i) y una imagen de amplitud (a_i) de la señal infrarroja, en las que para cada píxel se muestra el valor de distancia y la amplitud de la señal infrarroja recibida. Esta información puede proporcionarse en escala de grises, aunque habitualmente para facilitar la interpretación de los mapas de distancia se utilizan mapas de color en los que cada distancia se representa mediante un color diferente. La Figura 2.4 se muestra un ejemplo en el que se puede observar el funcionamiento de una cámara ToF y su proyección en dos dimensiones, observando la profundidad representada en una imagen mediante una escala de colores.

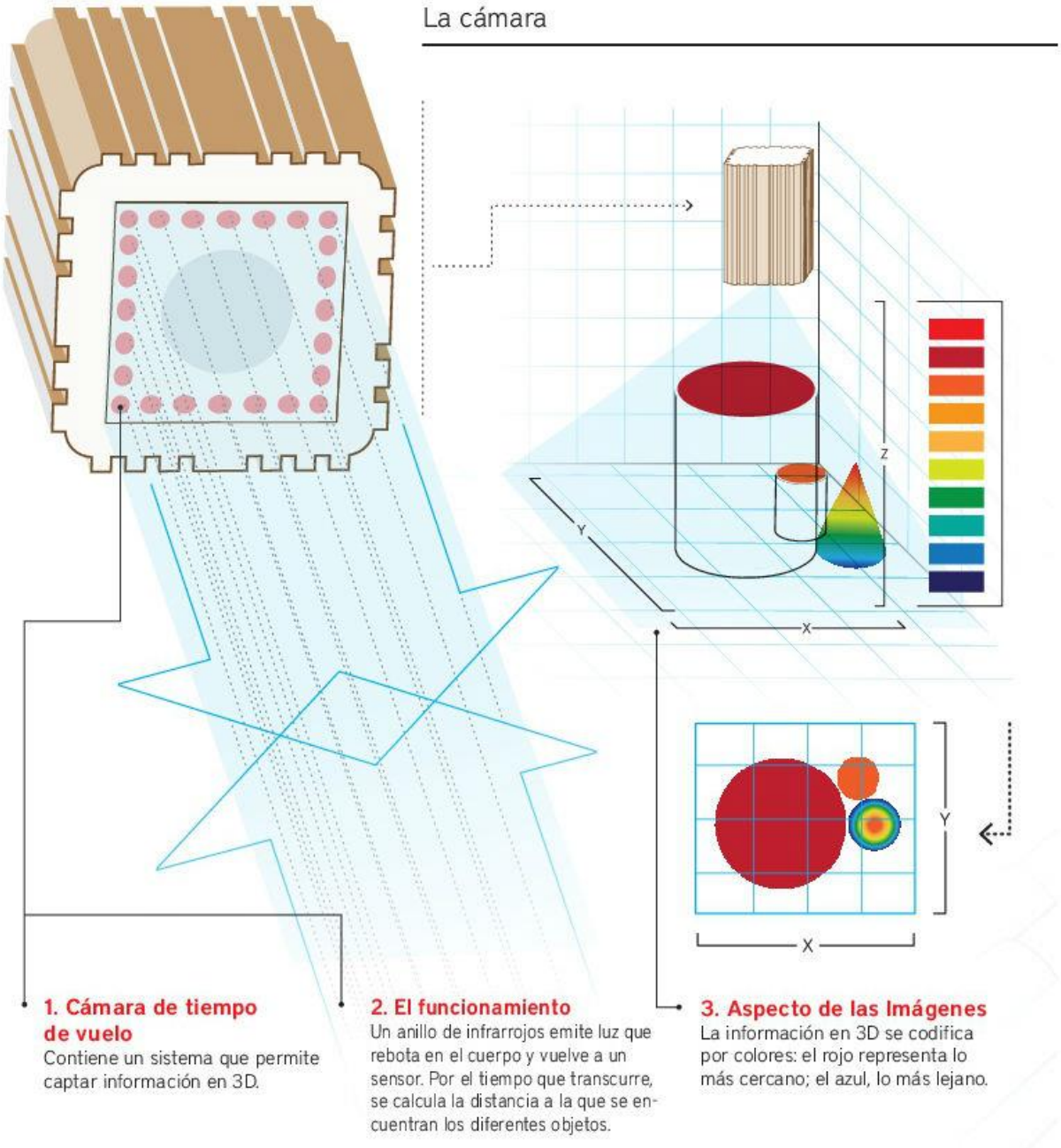

Figura 2.4 Esquema representación de imágenes de profundidad. [11]

Con lo anteriormente comentado, en la Figura 2.5, se presenta una imagen real de una cámara Kinect en posición cenital, la información de la profundidad está en una escala de grises.

Como se ha explicado anteriormente, las cámaras ToF emiten luz en el espectro infrarrojo, y calculan una estimación de la distancia al objetivo, d_i, basándose en el tiempo que tarda la luz emitida en ser reflejada y posteriormente recibida en el sensor de la cámara, que se mide de forma indirecta a partir de la diferencia de fase entre las señales emitida y recibida. Estos sensores tienen dos parámetros que determinan la precisión y robustez del cálculo de la distancia: el tiempo de integración (τ) y la frecuencia de modulación (f_{mod}) de la luz emitida.

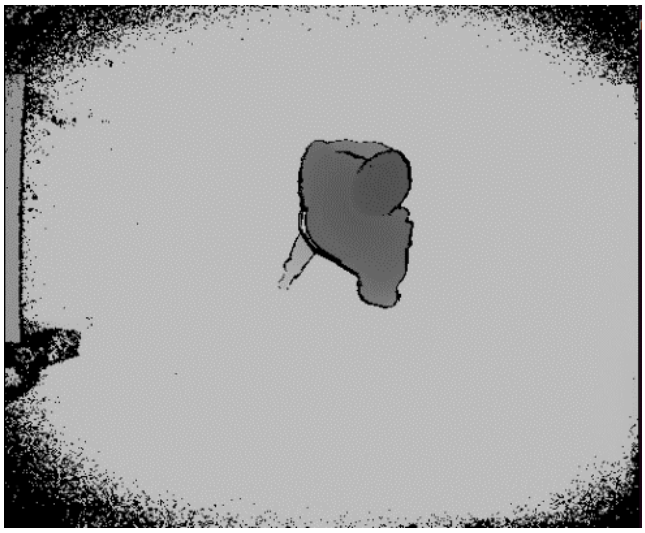

Figura 2.5 Representación de la profundidad obtenida con una cámara Kinect en posición cenital.

El tiempo de integración define el tiempo que requiere el sensor de la cámara para recibir la señal de luz reflejada por la superficie de los objetos que se han establecido como objetivos en la escena. Esto significa que, si el tiempo de integración es pequeño, la distancia a los objetos muy alejados no podrá ser medida, correctamente, puesto que la señal de luz no habrá tenido tiempo de retornar y excitar el sensor de la cámara. Además, por otro lado, si el tiempo de integración es grande, la cámara reduce su velocidad de adquisición y los mapas de profundidades pueden reflejar valores de distancia incorrectos.

Normalmente la frecuencia de modulación, f_{mod}, de la luz emitida es un parámetro fijo, pero cuando las características de la cámara permiten configurarlo, cambia la apariencia física de la forma sinusoidal que modela matemáticamente esa señal. Teniendo en cuenta que la distancia se mide de forma indirecta, a partir del desfase entre la señal emitida y la recibida utilizando la ecuación 2.3, y que el valor máximo de desfase que es posible tener sin ambigüedad es de 2π

radianes, la frecuencia de modulación determina el valor máximo de distancia que puede medirse sin ambigüedad (ya que no es posible diferenciar un desfase de 2πk + *n,* con *k*=1,2,… radianes de otro de *n* radianes).

Sustituyendo ϕ_i por 2π en la ecuación 2.4, se obtiene que la máxima distancia que puede medirse sin ambigüedad (d_{max}) es inversamente proporcional a la f_{mod}. Por ejemplo, para una cámara que emplea una frecuencia de modulación de 10Mhz, la distancia máxima a estimar en 15m.

$$d_{max} = \frac{\mathrm{c} \cdot 2\pi}{4\pi \cdot f_{mod}} = \frac{c}{2f_{mod}} \qquad 2.4$$

Para este trabajo, se ha elegido la cámara ToF Kinect v2, la cual será la parte del hardware destinada a la recolección de imágenes que actúan como información de entrada al sistema desarrollado. Se ha elegido esta cámara debido a que era la que se encontraba disponible para la realización de experimentos, sin embargo, el sistema desarrollado puedo trabajar sobre imágenes obtenidas con otras cámaras de profundidad basadas en tiempo de vuelo.

Las principales características de la Kinect2 utilizada se detallan a continuación:

- El campo de visión es de 70º en horizontal y 60º en vertical, esto permite poder detectar a varias personas dentro de un mismo campo de visión.
- Las imágenes de color se adquieren con una resolución de 1920 pixeles de ancho 1080 pixeles de alto, en formato full HD.
- Los mapas de profundidad tienen una resolución de 512 pixeles de ancho por 412 de alto.
- Frecuencia de modulación de 20Mhz, aplicando la ecuación 2.4, se obtiene una distancia máxima de 7.5 m.

Cabe destacar que, debido al principio de funcionamiento, la Kinect v2 es capaz de grabar imágenes de profundidad incluso en entornos no iluminados sin necesidad de una fuente de iluminación externa. Sin embargo, el uso de este tipo de sensores para la re-identificación de personas presenta problemas de precisión por los efectos de interferencia por multicamino [12] o los artefactos de movimiento [13] que deben tenerse en cuenta [14] [15] a la hora de procesar la información.

2.2. Filtro de Kalman discreto

En 1960, R.E. Kalman público un estudio en el cual, enunciaba una solución recursiva al problema de los filtros en sistemas discretos [16]. En dicho documento se enunciaba el filtro de Kalman, en honor a su creador, el cual tiene grandes aplicaciones en el campo de la navegación autónoma o asistida [17]. A continuación, se describen los fundamentos teóricos relacionados con el filtro de Kalman para la aplicación desarrollada en este TFG.

El filtro de Kalman intenta solucionar problema de la estimación del estado de un proceso controlado en tiempo discreto, el cual se rige por las ecuaciones en diferencias de estado (ecuación 2.5) y de medida (ecuación 2.6)

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \quad 2.5$$

$$z_k = Hx_k + v_k \quad 2.6$$

En las ecuaciones (2.5) y (2.6) las variables w_k (Ecuación 2.7) y v_k (Ecuación 2.8) representan el ruido de proceso y el ruido de medida, respectivamente. Asumiendo que ambas variables son independientes entre ellas y con distribuciones de probabilidad normales, pueden expresarse como se muestra en las ecuaciones 2.7 y 2.8.

$$p(w) \sim N(0, Q) \quad 2.7$$

$$p(v) \sim N(0, R) \quad 2.8$$

En la práctica, las matrices de covarianza del ruido de proceso (Q) y del ruido de medida (R), podrían cambiar con cada iteración del algoritmo, pero para este estudio se suponen constantes. Cada elemento de la matriz **A** (ecuación 2.5) se refiere al estado en el instante anterior *k*-1 respecto al instante actual *k*. La matriz **B** (ecuación 2.5) relaciona la entrada de control **U** y el estado *x* y la matriz **H** en la ecuación 2.6 relaciona el estado y la salida Z_k.

Definiendo $\hat{x}_k^- \in \Re^n$ como la estimación a priori del estado en el instante *k*, dado el conocimiento del proceso antes de la etapa *k*, y $\hat{x}_k \in \Re^n$ como la estimación a posteriori del estado en el paso *k*, cuya medición es Z_k, se pueden definir los errores de estimación a priori (ecuación 2.9) y a posteriori (ecuación 2.10)

$$e_k^- \equiv x_k - \hat{x}_k^- \quad 2.9$$

$$e_k \equiv x_k - \hat{x}_k \quad 2.10$$

Tras analizar el error de estimación a priori (Ecuación 2.9) y relacionarlo con las medidas estimadas obtenidas (Ecuación 2.10), se obtiene la ecuación (Ecuación 2.11), siendo K la ganancia. Es posible argumentar que para que la medida realizada y la estimación a priori de esta coincidan [18]. Esto ocurrirá cuando la ganancia sea cero o lo más cercano a ello.

$$x_k = \hat{x}_k^- + K(x_k - H\hat{x}_k^-) \quad 2.11$$

2.2.1. Algoritmo del filtro de Kalman (Discreto)

El filtro de Kalman realiza una estimación mediante un lazo de control realimentado, el filtro estima el estado del proceso y se corrige utilizando el dato real medido, siendo esta la realimentación anteriormente mencionada. Las ecuaciones que se muestran en la Figura 2.6 modelan el filtro de Kalman, se pueden agrupar en dos: aquellas relacionadas con el proceso de estimación ejecutado en cada instante de tiempo y aquellas que actualizan las medidas. El tiempo de actualización de las ecuaciones viene dado por la velocidad de muestreo al que se capturan los datos.

En cada iteración las ecuaciones se actualizan con los distintos valores actuales de la posición. Tras comprobar la estimación de la posición actual, calculada en el instante anterior, y el error cometido por la diferencia de los datos, obteniendo la estimación para el siguiente instante. Los resultados de las ecuaciones que estiman la posición actual serán los datos usados como realimentación del sistema, es decir, para incorporar una nueva medida en la estimación del instante actual previamente se debe haber realizado una estimación del instante anterior.

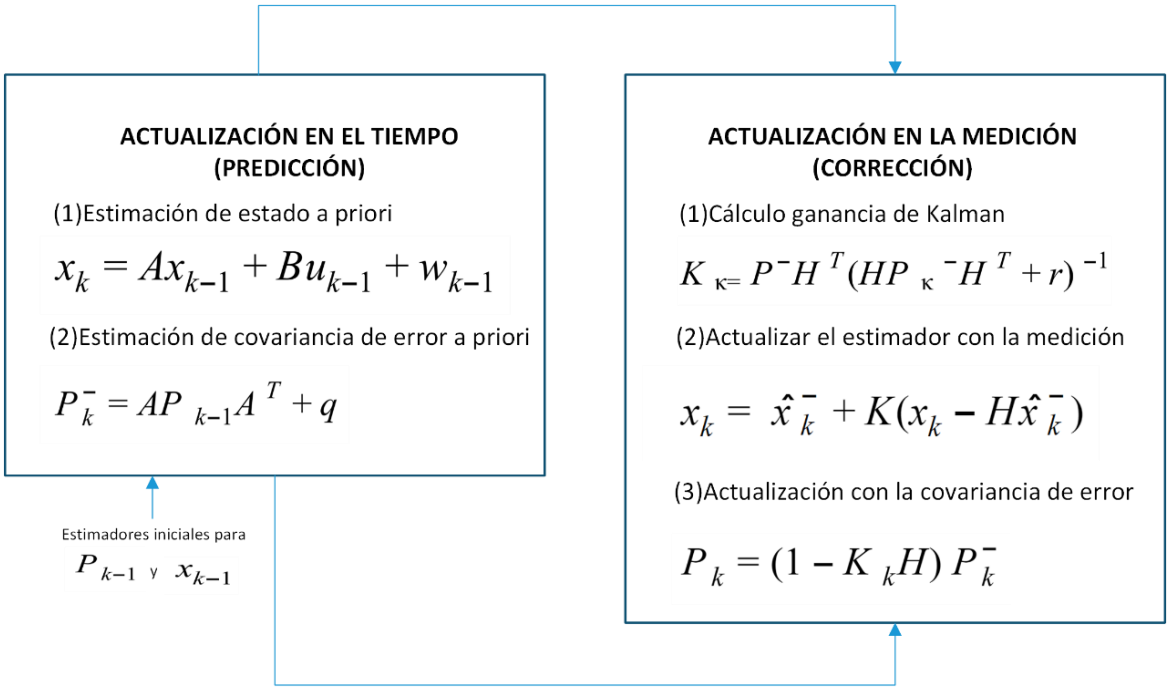

Figura 2.6 Diagrama de funcionamiento y ecuaciones asociadas al filtro de Kalman.

Por tanto, las ecuaciones de actualización de tiempo se pueden considerar ecuaciones de predicción y por otro lado las ecuaciones de actualización de la posición análogamente serian ecuaciones de corrección. Entonces se obtiene un sistema que similar a un modelo predictor-corrector para resolver problemas de posición.

Tras cada medición y su posterior actualización, el proceso se repite, siendo la medición actual el punto de partida para obtener la estimación a priori del instante siguiente. Esta naturaleza recursiva es una de las características más atractivas del filtro de Kalman, con esto se consigue que las implementaciones prácticas mucho más factibles que otros algoritmos con idéntica función, por ejemplo, un filtro de Wiener [19]. El cual está diseñado para funcionar con todos los datos directamente en cada estimación. El filtro de Kalman es considerado recursivo para la medida actual sobre todas las medidas pasadas.

2.2.2. Parámetros del filtro de Kalman

La medición de la covarianza de error de medición **R** es generalmente práctica porque es necesario poder medir el estado del filtro durante la operación de este. La determinación de la covarianza de ruido de proceso **Q** es generalmente más difícil ya que normalmente no se tiene la capacidad de observar directamente el proceso que se está estimando. A veces la mejor forma de mejorar esto es utilizando un modelo de proceso relativamente simple, para producir resultados aceptables.

En cualquier caso, incluso tras elegir los parámetros siguiendo una estrategia fundamentada, muchas veces se puede obtener un rendimiento superior del filtro, estadísticamente hablando, ajustando los parámetros **Q** y **R** utilizando un filtro de control. Es decir, realizando una sintonización con otro filtro de Kalman, distinto al sintonizado, en un proceso y entorno generalmente conocido para poder extraer así los datos [20].

En resumen, se observa que bajo condiciones donde **Q** y **R** son de hecho constantes, la estimación de la covarianza de P_K y la ganancia de K_K de Kalman se estabilizarán rápidamente y luego se mantendrán contantes. Si este es el caso, estos parámetros se pueden pre-calcular ya sea ejecutando el filtro del control o determinando el valor de estado estable P_K [20].

Sin embargo, es frecuente que el error de medición no permanezca constante. Por ejemplo, cuando se realizan medidas de objetivos estáticos respecto al punto de medida, el ruido en las mediciones de los objetivos más cercanos será menor que en los objetivos lejanos. También puede ocurrir que el ruido del proceso cambie dinámicamente durante la operación del filtro, teniendo este que ajustarse de forma dinámica. Por ejemplo, en el caso de rastrear un objetivo en un entorno virtual 3D y con un movimiento en 3D, sería posible reducir la magnitud de dicho error si el objetivo parece realizar un movimiento lentamente, por el contrario, aumentaría de magnitud si la dinámica comienza a cambiar rápidamente.

2.3. Reconocimiento de acciones

El reconocimiento de acciones es un área de investigación en expansión debido a las potenciales aplicaciones. Existen distintas técnicas para esto, basadas en la definición de un modelo del cuerpo humano o en las características extraídas directamente de un archivo de vídeo.

Este último caso se centra en el análisis del movimiento sobre su patrón temporal o sobre la forma del sujeto [21]. Las diferentes alternativas propuestas en la literatura científica pueden agruparse en función de las técnicas que utilizan en:

- Aquellos que aproximan la detección de la acción como un problema de emparejamiento de patrones, en los que se extrae un patrón de la secuencia de imágenes que conforma el video, y se compara con el prototipo almacenado. Algunos de los clasificadores más usados son el vecino más cercano [22], máquinas de vectores soporte [23] o las redes neuronales artificiales [24].
- Los modelos de espacios de estados, que definen un espacio de estados en el que cada punto de este espacio representa un evento (pose, movimiento…) de la acción, proporcionando un modo compacto de describirla. Estos modelos pueden ser deterministas, como las máquinas de estados finitos [25], que conciben las acciones como procesos totalmente observables en el que el conjunto de estado es conocido. A cada estado se le asocia una probabilidad, siendo la probabilidad conjunta de todos los estados que conforman el modelo de cada acción, el criterio usado en la clasificación.

Se ha decidido implementar la primera opción, de las anteriormente documentadas, debido a la información disponible. Las escenas al ser capturadas desde una posición cenital resultan insuficientes para conformar un modelo de espacios de estados, pero si aportan los datos necesarios para poder obtener la detección de la acción usando reglas heurísticas.

2.4. Principios de diseño SOLID para desarrollo de software

El trabajo ha sido desarrollado siguiendo los principios SOLID, que es un acrónimo inventado por Robert C. Martin para establecer los cinco principios básicos de la programación orientada a objetos y diseño [5].

Este acrónimo, cuyas partes se detallan a continuación, está relacionado con los patrones de diseño.

***S-Responsabilidad única (Single responsibility)*:**

Este principio trata de destinar cada clase a una finalidad sencilla y concreta.

O-Abierto/Cerrado (Open/Closed)

El diseño debe ser abierto para poderse extender, pero cerrado para poderse modificar. Para conseguir este principio hay que tener muy claro cómo va a funcionar la aplicación, por donde se puede extender y cómo van a interactuar las clases.

L-Sustitución Liskov (Liskov substitution)

Al crear clases derivadas no se debe reimplementar métodos que hagan que los métodos de la clase base no funcionasen si se tratasen como un objeto de esa clase base.

I-Segregación de interfaz (Interface segregation)

Este principio fue formulado por Robert C. Martin y trata de algo parecido al primer principio. Cuando se definen interfaces estos deben ser específicos a una finalidad concreta.

D-Inversión de dependencias (Dependency inversion)

También fue definido por Robert C. Martin. El objetivo de este principio conseguir desacoplar las clases. En todo diseño siempre debe existir un acoplamiento, pero hay que evitarlo en la medida de lo posible. Un sistema no acoplado no hace nada, pero un sistema altamente acoplado es muy difícil de mantener.

El objetivo de este principio es el uso de abstracciones para conseguir que una clase interactúe con otras clases sin que las conozca directamente.

Hay múltiples razones para seguir un buen diseño de programación, una de ellas es abarcar la fase de mantenimiento de una manera más legible y sencilla, así como conseguir crear nuevas funcionalidades sin tener que modificar en gran medida código antiguo. Durante la fase de depuración solucionar distintos errores, al tener el código compartimentado, se puede realizar implementando test unitarios bien definidos.

2.5. Herramientas software

En lo referente a los sistemas operativos, quedan atrás ya los ineficientes sistemas monolíticos. Con el fin de aprovechar los recursos *hardware* de una máquina, se han creado distintos proyectos que en mayor o menor medida implementan esta idea.

Docker es uno de estos proyectos tiene como paradigma principal un contenedor como servicio (*CaaS: Container as a Service*). Ampliamente implementado en la industria, en servicios con una alta demanda de usuarios como Spotify, Twitter o BBVA entre otros [26].

Existe una necesidad de conocer el estado de los procesos, recursos hardware y distintas estadísticas relacionadas con el correcto funcionamiento de los sistemas. Por razones de disponibilidad, es recomendable que dichas estadísticas sean almacenadas fuera de los propios recursos *hardware*, con un software ligero y compatible con sistemas UNIX.

2.5.1. Docker

Docker [27] es un proyecto, de carácter *open source*, creado por Docker Inc.; la cual es la principal promotora de este proyecto y de las herramientas que conforman la suite completa. Se puede definir Docker como, el software de TI (Tecnologías de la Información), siendo una tecnología de creación de contenedores que permite la creación y el uso de contenedores Linux. Cuenta con una comunidad *open source* que trabaja para mejorar la tecnología con el fin de beneficiar a todos los usuarios de forma gratuita, así como con el desarrollo de la empresa Docker Inc., respaldando las tecnologías mejoradas y reforzadas para clientes empresariales.

Todo esto hace que con Docker se pueda considerar los contenedores como máquinas virtuales extremadamente livianas y modulares, además de obtener la flexibilidad de creación, implementación, copia y traslado de dichos contenedores de un entorno a otro.

Capítulo 3.

Desarrollo

A lo largo de este capítulo se presenta el desarrollo del sistema implementado en este TFG, cuyas etapas se muestran en el diagrama de bloques de la Figura 3.1. En este diagrama se incluye desde la adquisición de imágenes por la cámara ToF hasta el fichero de salida en el que se almacenan las mediciones del sistema. Se describen en detalle las siguientes etapas:

- Adquisición de la información profundidad de la escena utilizando una cámara Kinect v2 en posición cenital.
- Identificación del número y obtención de la posición de las personas presentes en la escena.
- Seguimiento de un número variable de personas mediante un banco de filtros de Kalman:
    - Estimación de las posiciones de las personas presentes en la anterior escena.
    - Asociación de las estimaciones con las lecturas reales.
    - Corrección de las estimaciones, incluyendo en este punto el estudio de la aparición o desaparición de una persona en la escena para los instantes posteriores.
- Detección de las acciones realizadas por las personas presentes en la escena, a partir de las predicciones estimadas. Con especial énfasis en las caídas.

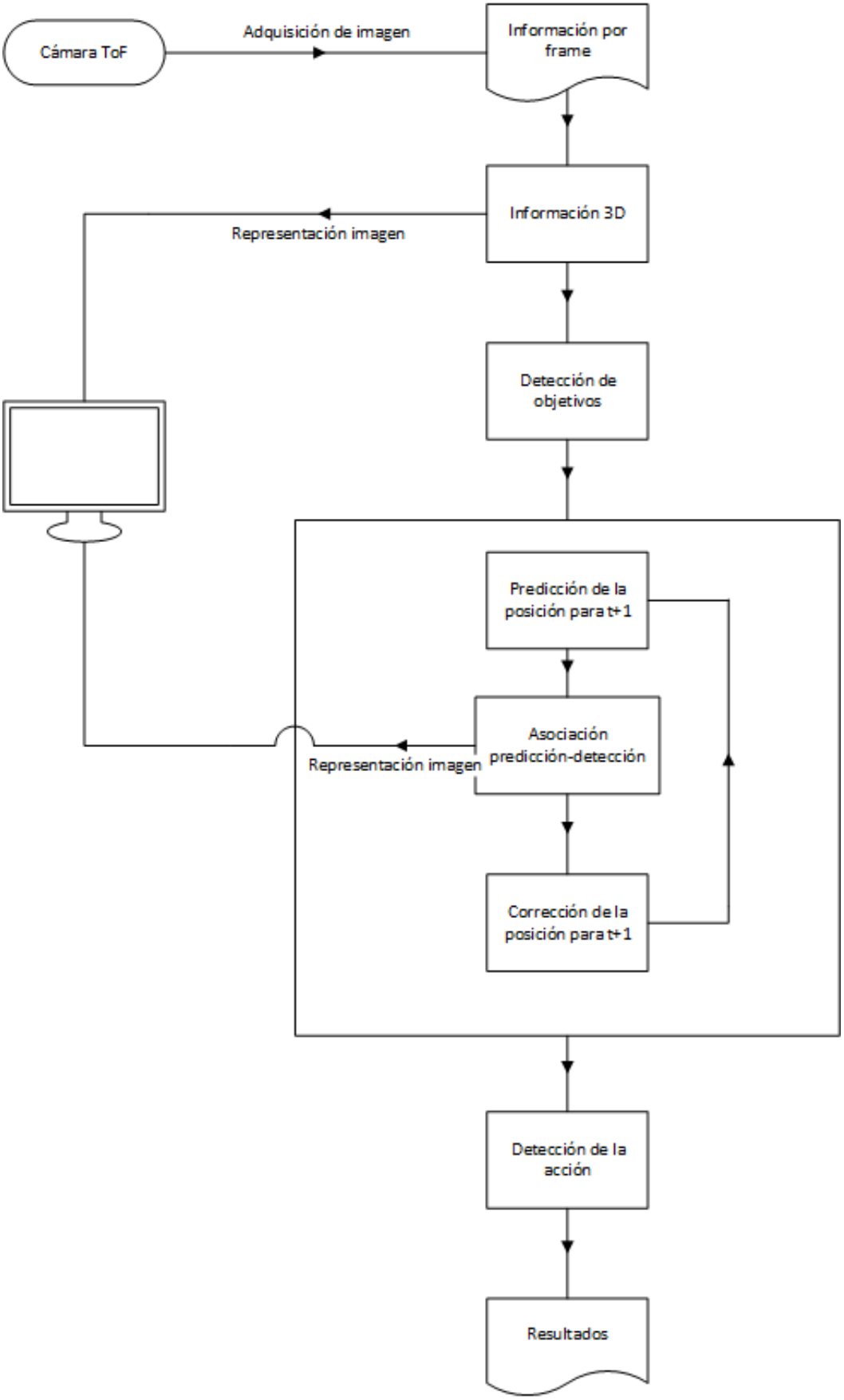

Figura 3.1 Diagrama de bloques del trabajo desarrollado.

3.1. Adquisición y representación de información

Como se ha comentado, la adquisición de datos del sistema se realiza con una cámara Kinect v2, a una velocidad de muestreo de 30Hz, dando por tanto una imagen, a partir de ahora llamada escena, cada aproximadamente 33 ms. La cámara proporciona un mapa de profundidad que contiene la distancia a cada punto de la escena, (x, y) sobre el plano imagen y el valor de la profundidad (eje Z), de 424 x 512 pixeles. La Figura 3.2 muestra una representación, en escala de grises, de la profundidad captada en una imagen de ejemplo, aquellos puntos más claros se encuentran a una distancia mayor de la cámara que los oscuros. Dicha cámara se encuentra en posición cenital a 3,4 metros del suelo en un entorno interior, la posición de la cámara es estática, por tanto, el fondo de la imagen no varía.

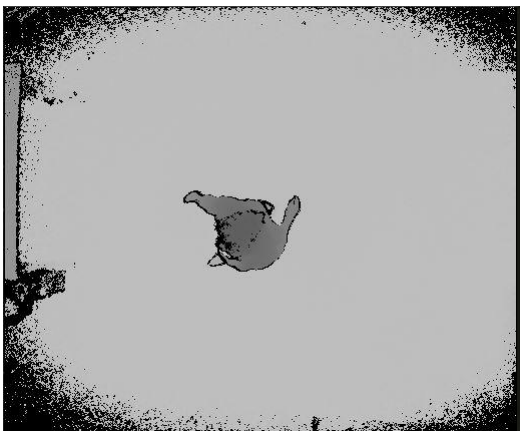

Figura 3.2 Representación de la profundidad obtenida mediante una cámara ToF.

Las imágenes adquiridas se almacenan en un fichero codificado, con formato de 16 bits, donde se encuentra organizado de forma que la profundidad de cada pixel se graba en el documento en orden de posición de fila siendo el primero el que corresponde al pixel que se encuentra en la esquina superior izquierda de la primera *escena*. La información del video se encuentra almacenada en un archivo, en la que cada elemento corresponde a un pixel del video, cuya posición en la escena es (*x,y),* dicha escena se encuentra en la posición *m* dentro del vídeo. De esta estructura se infieren las ecuaciones 3.1 y 3.2 para poder definir la información de cualquier pixel almacenado en el archivo.

$$elemento_N = frame + pixel_x + pixel_y \quad (3.1)$$

$$elemento_N = \left\lfloor \frac{N}{512 * 424} \right\rfloor + \left\lfloor \frac{N}{512} - \frac{N}{512 * 424} \right\rfloor + (N \ mod \ 512) \quad (3.2)$$

A partir del fichero de datos se puede decodificar el video como un conjunto de imágenes equiespaciadas temporalmente en fragmentos de 33 ms. Por tanto, para representarlo, basta con cargar el mapa de pixeles correspondiente a cada *escena* cada intervalo de tiempo con el que se muestrearon las imágenes.

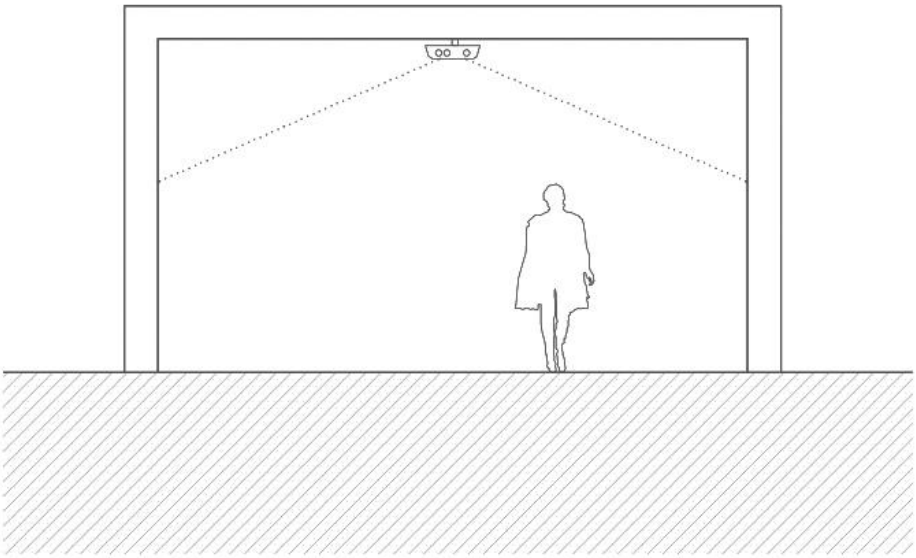

Figura 3.3 Representación esquemática de la cámara en posición cenital.

Para la representación de imágenes hay que tener en cuenta cómo se codifica la información que se almacena de cada pixel: la profundidad se mide tomando como punto de referencia la propia cámara, como es de esperar. Al encontrarse la cámara en posición cenital, representada en la Figura 3.3 y apuntando al suelo, un objeto situado en el mismo tendrá una medida inferior que el propio suelo. La Figura 3.4 representa la imagen siguiendo un código de colores (Tabla 3.1) para los distintos valores de profundidad, normalizados respecto a la altura de la cámara respecto al suelo. Dicho código de colores es arbitrario, únicamente busca crear un contraste bien definido para poder interpretar de una forma visual la información almacenada en la imagen.

Intervalo (normalizado a 3,4m)	Color
$valor_{profundidad} = 0$	Rojo
$0,3 \leq valor_{profundidad} < 0$	Negro
$0,4 \leq valor_{profundidad} < 0,3$	Verde Oscuro
$0,5 \leq valor_{profundidad} < 0,4$	Verde Claro
$0,7 \leq valor_{profundidad} < 0,5$	Blanco
$valor_{profundidad} > 0,7$	Azul

Tabla 3.1: Código de colores en función de la profundidad.

Figura 3.4 Representación imagen asignando un color a cada rango de valores de profundidad.

3.2. Detección de personas

Tras extraer la información del archivo, es necesario identificar las personas, en los que se centra este trabajo. Bajo la premisa que en primera instancia los objetivos son personas adultas con una estatura media, se ha realizado un algoritmo que analiza la información por cada *escena*, buscando una persona en movimiento y obteniendo la estatura media de esta.

Para poder identificar si hay una persona en una imagen, los pasos seguidos son los siguientes:

1. Se realiza una resta de fondo con respecto a la escena anterior, obteniendo así el contenido dinámico de la escena.
2. Se estudia todos los puntos dinámicos, considerando una persona válida aquella que supere una superficie mínima de 15 *pixeles* por 15 *pixeles*.
3. En el caso de encontrar alguna coincidencia en el punto anterior, se obtiene todos los puntos que conforman la persona utilizando la información obtenida en el primer punto.
4. Se genera una superficie cuadrada que contiene a la persona y se almacenan dichos datos para poder usarlos en la representación.
5. Se calcula la estatura media de la persona, como el sumatorio de las alturas de los puntos asociados a la persona.
6. Finalizado esto, se vuelve al primer paso hasta finalizar de analizar la escena.

Se ha decidido emplear un detector de máximos, en lugar de un clasificador debido a que la detección de personas no era parte de los objetivos del trabajo., sin embargo, dada la estructura modular del trabajo desarrollado, es posible incorporar el clasificador de forma sencilla, sustituyendo el detector de máximos.

Figura 3.5 Imagen modelo de fondo.

La Figura 3.5 se observa un ejemplo del modelo de fondo, una imagen en la que aparece solo los objetos estáticos que conforman la escena. La información contenida en este modelo de fondo será utilizada para realizar una resta de fondo con cada imagen del video con el fin solventar problemas de falsos positivos.

Las escenas de estos videos en los que no aparece ningún objetivo válido se han utilizado para crear el modelo de fondo, y analizar los falsos positivos relacionados con el ruido por los efectos de borde. Se ha creado un modelo de fondo como resultado de promediar los valores asociados a cada pixel en varias imágenes, en estas que no hay ningún objetivo válido en pantalla.

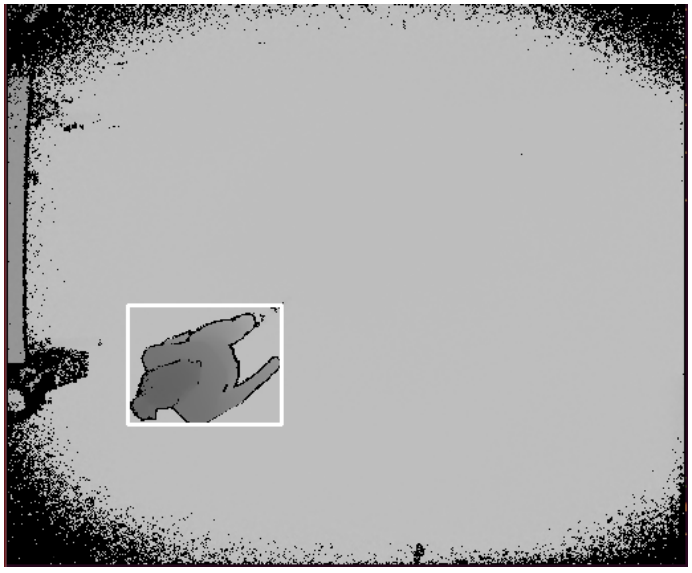

Figura 3.6 Imagen con una persona identificada.

La Figura 3.6 corresponde a una escena en la que aparece una persona, la cual se ha identificado siguiendo los pasos anteriormente mencionados. El contorno varía por cada escena en función del área detectada.

Cabe destacar que, una de las bondades al desarrollar el código bajo los preceptos SOLID, es la posibilidad de intercambiar fácilmente este detector de máximos por un clasificador de personas que cumpla las mismas funciones. Pudiendo escoger otra cámara de la cual se extraiga la información necesaria o incluso modificar la técnica para detectar personas. Los único que se debiera asegurar es que el desarrollo a incluir tenga una estructura, a nivel de clase y métodos, similar, conservando en cada método la misma estructura en el flujo de salida.

3.3. Seguimiento de personas

Con la información obtenida tras el proceso de detección de personas en la escena, se va a realizar el seguimiento de estas. Esta etapa de seguimiento está conformada por la predicción de las posiciones, asociación de dichas predicciones a las personas presentes en la escena, corrección de los errores cometidos durante la predicción respecto a las posiciones reales y finalmente una etapa de validación del estado de los filtros. Debido a que se realiza el seguimiento de un número variable de personas, es necesario contar con mecanismos que permitan tanto la creación de nuevos filtros si aparece una nueva persona, como la eliminación de estos, en caso de que una persona abandone la escena.

3.3.1. Etapa de predicción

En lo concerniente a la etapa de predicción de personas, el tiempo concurrido entre ambas escenas es considerado una constante para el filtro de Kalman, para así poder realizar una estimación de la posición de cada persona en el siguiente instante de medición. Para modelar el movimiento de las personas en el filtro de Kalman, se emplea la ecuación 3.3, en la que se muestra la matriz de transición. Se puede observar como el vector de estado está formado por 6 componentes: las tres primeras corresponden a las coordenadas 3D de la persona (x, y, z) y las tres últimas son las componentes de velocidad en los ejes x, y, z respectivamente, mientras que T representa el tiempo transcurrido entre escenas. Por otro lado, la expresión 3.4 muestra la ecuación de medida del filtro, en la que se puede observar que únicamente se miden las coordenadas 3D de la persona, ya que la medida de velocidad en las imágenes es demasiado ruidosa para la aplicación desarrollada.

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \\ z_{n+1} \\ v_{x_{n+1}} \\ v_{y_{n+1}} \\ v_{z_{n+1}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ z_n \\ v_{x_n} \\ v_{y_n} \\ v_{z_n} \end{bmatrix} \quad 3.3$$

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \\ z_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{n+1} \\ y_{n+1} \\ z_{n+1} \\ v_{x_{n+1}} \\ v_{y_{n+1}} \\ v_{z_{n+1}} \end{bmatrix} \quad 3.4$$

Tras ejecutar las predicciones de todos los filtros que se encuentren activos o en espera, la información almacenada en la matriz de estado contendrá las estimaciones de donde es posible que se encuentren los objetivos en la siguiente *escena*. Al realizar una estimación de un filtro, este debe considerarse para la etapa de asociación. Las estimaciones son almacenadas en un vector dedicado exclusivo a cada filtro, para su posterior asociación con las medidas extraídas de cada escena.

Por tanto, a excepción de la primera *escena*, se tiene información de las posibles localizaciones de los objetivos en el caso de que los hubiese, la siguiente acción a realizar es asociar las predicciones con las mediciones obtenidas.

3.3.2. Etapa de asociación de medidas

Para realizar la asociación de cada estimación con su medida, se ha implementado el algoritmo del vecino más cercano (*Nearest Neighbour*). Es decir, a una estimación le corresponderá la medida de la persona más cercana, siempre y cuando esta se encuentre en el radio de acción de la estimación en el que se considera que puede encontrarse la persona que está siendo seguida.

Cuando un filtro realiza una estimación, activa un registro que lo identifica para que sea considerado en esta etapa de asociación. Si se produce una correspondencia entre una medida y un filtro, el registro asociado a dicho filtro se desactiva, para no ser considerado por siguientes medidas de la misma escena.

Al estudiar el caso particular mostrado en la Figura 3.7, existen dos objetivos los cuales se aproximan con vectores de dirección perpendiculares, a velocidad constante y similar. Se capturan tres escenas asociadas al acercamiento, al cruce de los objetivos y al distanciamiento de estos. Durante la fase de acercamiento, la diferenciación entre los objetivos y sus predicciones correspondientes está claro; en el cruce, los objetivos se encuentran tan próximos que el error por una mala asociación de sus predicciones es mínimo y, por último, en la fase de distanciamiento, los objetivos empiezan a separarse y sus predicciones son fácilmente diferenciables. El porqué de este ejemplo es para ilustrar porque se asocian objetivos a una predicción y no se mantiene una relación constante de un filtro con un objetivo.

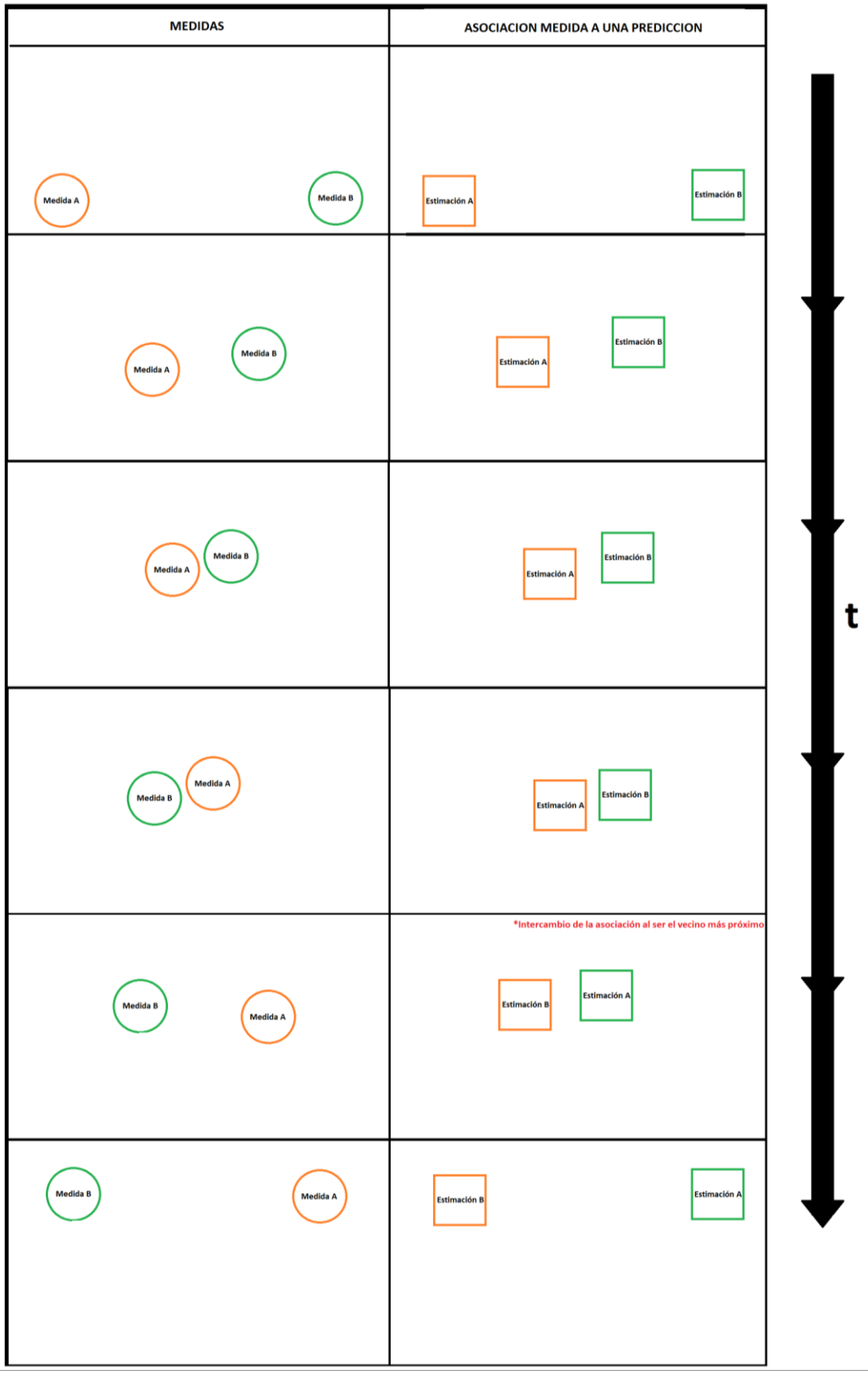

Figura 3.7 Representación de la secuencia de dos objetivos con rumbo de colisión.

En el caso contrario, es decir, asociar las predicciones a una medida el error cometido en la fase de distanciamiento sería inaceptable, tras esto, en las siguientes iteraciones, el filtro iría ajustándose, generando predicciones que redujeran el fallo cometido respecto a la posición del objetivo asociado, llegando tras ciertas escenas a dar resultados similares a los asociados antes del cruce.

3.3.3. Etapa de corrección

Tras asociar las medidas a sus estimaciones, el siguiente paso es actualizar las posiciones medidas de las personas. Esto tiene especial importancia al usar la medida como punto de origen para la estimación de la siguiente *escena* y sirve para calcular el error cometido entre la predicción y la medida, y asi reajustar las siguientes predicciones en función del error cometido.

Se ejecutan las correcciones sobre todo filtro no inactivo, bien tenga una medida asociada o por el contrario tras el proceso de asociación no hubiera obtenido medida alguna. Los parámetros de entrada para la actualización del filtro, el vector de medidas (x,y,z) asociado a este , se introducen en la matriz de corrección relacionada con cada filtro. Finalizado el proceso de asociación de las estimaciones con las medidas más probables, se puede dar el caso que un filtro no tenga ninguna medida asociada pero que aún siga considerándose activo, en ambos casos se debe realizar una corrección de estos. Si el filtro tiene una medida asociada, esta matriz de medidas será rellenada con los datos de dicha asociación. Por el contrario, si el filtro no tiene ninguna medida asociada, en la matriz se incluirá la estimación del filtro para realizar la corrección.

3.3.4. Creación y destrucción de los filtros

En lo relacionado con la creación y destrucción de filtro se ha implementado un vector asociado a cada filtro, en el que se asocia un estado siguiendo una filosofía tri-estado. Los distintos estados en los que se puede encontrar el filtro son: activo, en espera e inactivo.

Cuando un filtro se encuentra en estado activo, es porque tiene una correspondencia con la medida de una persona en escena, y en espera cuando dicha correspondencia no se puede considerar completa por falta de medidas consecutivas de esa persona. Por tanto, los distintos estados se definen:

- Estado Activo: Filtro activo, con medidas válidas.
- Estado En espera: Filtro activo, con medidas no válidas.
- Estado Inactivo: Filtro inactivo.

Las transiciones entre estados se realizan en base a la asociación de la estimación del filtro con una medición, siendo suficiente para considerar inactivo un filtro la no asociación de mediciones durante cinco escenas consecutivas, mientras que, la transición a activo se hace forma simétrica. En los periodos en los que se cuentan el número de escenas consecutivas para estas transiciones, los filtros se encuentran en un estado en espera. Durante este estado los resultados consecuencia de las estimaciones asociadas no se tienen en cuenta para el cálculo de acciones. Puede ocurrir transiciones activo-en espera-activo, y también el caso contrario, esto se debe a la casuística en la que durante un periodo reducido de escenas el filtro no tiene ninguna medida válida asociada pero antes de llegar a su destrucción se retoma la secuencia de medidas adecuadas.

Al entrar una persona en escena, el proceso de asociación de objetivo predicción no converge hasta pasado cierto tiempo. Empíricamente se ha comprobado que con 5 detecciones consecutivas de la misma persona son suficientes para considerar a dicha persona válida. Esto podría provocar que el sistema detecte dos usuarios distintos durante este periodo, ya que al utilizar como criterio el algoritmo del vecino más próximo unido entorno a un umbral de acción, cuando una persona entra en escena y se realiza la primera predicción, la distancia entre ambas es demasiado extensa como para que se pueda asociar a la misma persona. Durante las siguientes escenas las distancia entre las predicciones resultantes y la posición real disminuyen, y el sistema converge. Para evitar estas múltiples detecciones hasta la convergencia, se define el estado del filtro en espera, se encuentra activo pero la información que contiene no es fiable.

Cada filtro tiene un contador independiente para modificar el estado en el que se encuentra, tal como se puede observar en la Figura 3.8, en la operativa pueden ocurrir las siguientes transiciones:

- Activo –> Activo: Hay una medida asociada pero el contador ya se encuentra en su valor máximo, se realiza la corrección con la medida de la persona asociada.
- Activo –> En espera: No hay una medida asociada el contador se decrementa y es distinto de su valor máximo, se realiza la corrección con la medida de la persona asociada.

- En espera –> En espera:
    - Hay una medida asociada, el contador se incrementa y se realiza la corrección con la medida de la persona asociada.
    - No hay una medida asociada, el contador se decrementa y se realiza la corrección con la medida de la persona asociada.
- En espera –> Inactivo: El filtro no tiene ninguna medida asociada y se corrige con la estimación propia, al decrementar el contador este llega a su valor mínimo. Esto significa que en la siguiente iteración dicho filtro no aparecerá en la lista de filtros activos, considerándose inactivo.
- Inactivo –> Inactivo: No hay ninguna medida asociada y el contador se encuentra en su valor mínimo, nada que hacer.
- Inactivo –> En espera: Una persona detectada, tras el proceso de emparejamiento de filtros activos con sus posibles objetivos, no tiene ningún filtro asociado. Se inicializa un filtro inactivo y se incrementa el contador asociado, por tanto, en la siguiente iteración el proceso de emparejamiento incluirá dicho filtro en la lista de candidatos.
- En espera –> Activo: Hay una medida asociada, el contador se incrementa y alcanza su valor máximo, se realiza la corrección con la medida de la persona asociada.

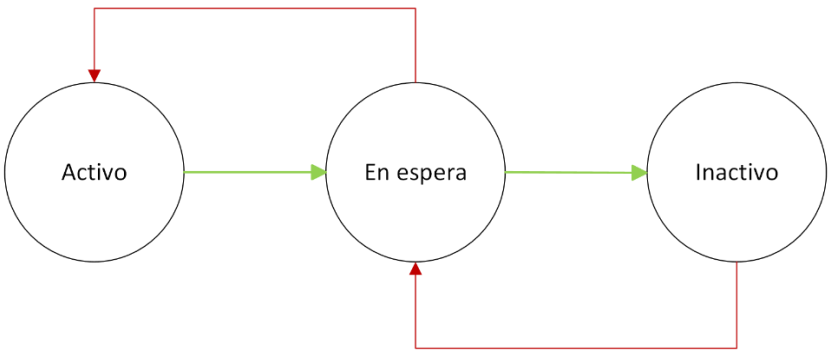

Figura 3.8 Diagrama de estado del filtro de Kalman.

3.4. Reconocimiento de acciones y detección de caídas

La detección de una acción detectada se hace con la información obtenida tras varias imágenes consecutivas. Una vez obtenidas las predicciones de las personas en cada escena, se procede a reconocer la acción para aquellas que sean consideradas como medidas asociadas a un filtro en estado activo. No se realiza el reconocimiento de acciones sobre los filtros en estado de espera al no tener la información suficiente para poder obtener resultados fiables.

Las acciones que se van a estudiar son aquellas asociadas al movimiento en cada uno de los ejes cartesianos; debido a la posición cenital de la cámara, la información obtenida es insuficiente para poder detectar acciones más complejas como una persona sobre un monopatín o bailando Funk.

La Figura 3.9 y la Figura 3.10 son capturas de dos escenas no consecutivas de un mismo video, en las que aparece la misma persona con el identificador de la predicción de Kalman, sobre las cuales se calcula la velocidad. Al extraer la posición de la predicción de la Figura 3.10 y la posición de la predicción de la Figura 3.9, se puede obtener el vector de movimiento y la distancia recorrida entre ambas escenas. Con esta información y el tiempo transcurrido entre escenas, se calcula la velocidad media que lleva la persona y se estipula la acción (correr, andar, estar parado o caer) que se realiza en ese momento.

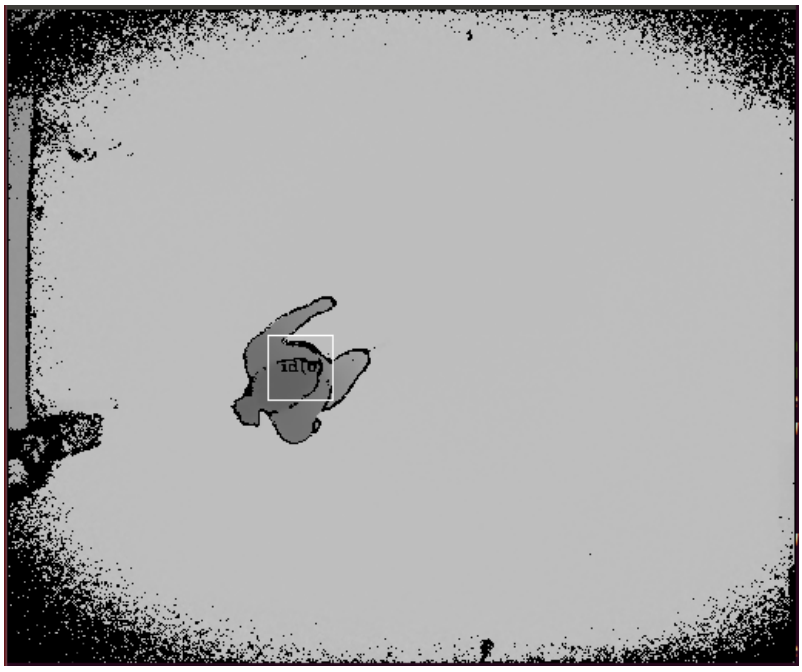

Figura 3.9 Escena con la predicción de una persona identificada en el instante t1.

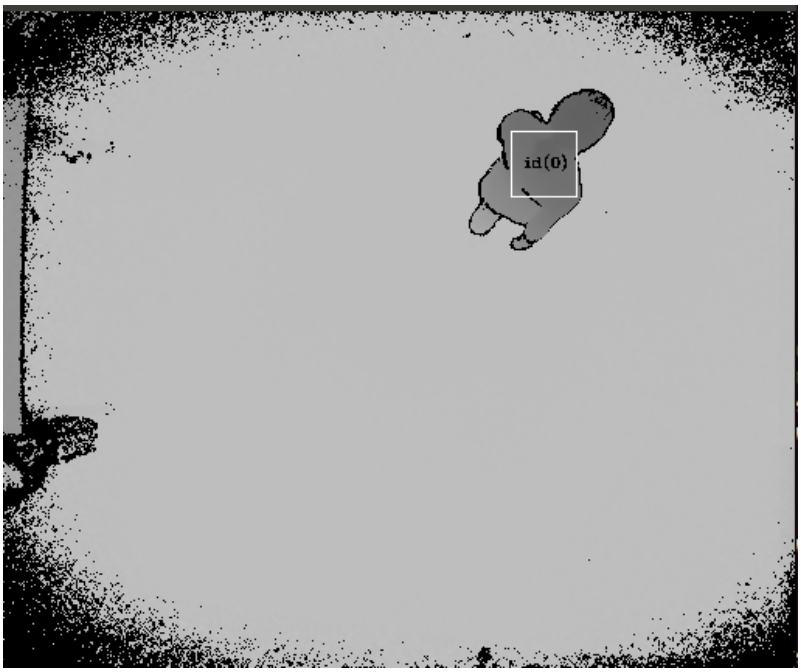

Figura 3.10 Escena con la predicción de una persona identificada en el instante t2.

El método para obtener dicha acción es siguiendo patrones heurísticos, para obtener los valores de referencia se ha medido a distintas personas realizando dichas acciones, y se ha seleccionado un conjunto de videos ejecutados por personas diferentes para el proceso de extracción de resultados. No se ha escogido a una única persona realizando todas las acciones posibles para evitar que los datos estén condicionados a la altura de dicha persona y rasgos propios de su cadencia natural al moverse. Se han estudiado distintos videos, en lo que aparecía una persona realizando una acción, con el fin de establecer unos parámetros heurísticos con diferentes condiciones y umbrales en función de la velocidad, tanto en el plano XY, como en el eje Z.

Dichas acciones se pueden dividir en dos grupos utilizando como criterio las coordenadas que varían, por un lado, el plano XY y por otro el eje Z.

Todas estas acciones solo pueden ser detectadas si se cuenta con una ordenada de imágenes, ya que es en la evolución de las distintas imágenes asociadas a un patrón temporal donde se puede estudiar las variaciones en los distintos ejes.

- Para la clasificación de las diferentes acciones a detectar, se plantea un sistema heurístico a partir de la información de la velocidad y dirección de los cambios en la posición de las personas, considerándose los siguientes estados: Estado Parado – Variación lenta, casi inexistente en los tres ejes.

- Estado Andando – Variación media en los ejes X-Y, y variación lenta o nula en el eje Z.
- Estado Corriendo – Variación rápida en los ejes X-Y.
- Estado cayendo – Variación abrupta de naturaleza negativa, indicador de un descenso rápido en las coordenadas del eje Z.

En cada escena se estudia la velocidad de las estimaciones de los filtros con estado activo y en espera, aunque solo se consideran para el reconocimiento de acciones la información de los filtros activos, aun cuando no tienen una imagen asociada en la escena.

Para detectar las caídas de forma robusta, además de realizar la media de la altura como se ha explicado anteriormente, se realiza un análisis de la evolución de la posición y velocidad en la coordenada Z, consiguiendo así resultados más estables.

3.5. Aplicación del patrón de diseño SOLID

En el capítulo del estudio teórico se ha visto el patrón de diseño SOLID. En este apartado se detalla cómo han sido aplicados los preceptos de este patrón en la aplicación desarrollada en este TFG:

- **S –** Métodos concretos, que realizan funciones específicas, sin caer en la repetición innecesaria de código.
- **O** – Aquella información heredada por una clase o método, se referencia mediante el flujo y no mediante variables globales. Las variables necesarias para el buen funcionamiento de una clase, permanecerán privadas, y en el caso de ser necesario se deben implementar métodos de modificación y/o consulta de dichas variables.
- **L** - Abstracción de los métodos, mediante una parametrización mediante el flujo de entrada.
- **I** – Los métodos son únicos en lo que se refiere a su función.
- **D** – Las clases son tratadas identidades opacas, un flujo de entrada genera un flujo de salida. Reduciendo las señales e interacciones con el resto de clases a lo imprescindible.

3.6. Almacenamiento y representación de resultados

El sistema almacena en ficheros tres fuentes de información, un log asociado a la ejecución del sistema, las acciones detectadas por escena y la velocidad de las medidas.

Para el *log* se ha creado una clase específica, con el fin de darle uniformidad al mismo. El *log* del trabajo se almacena en texto plano y es independiente por cada clase, indicando dos niveles de criticidad, *info* y *debug*. El nivel por defecto es *Debug* y se puede configurar en el constructor de cada clase, no se implementa ningún método en la clase para modificarlo.

Tanto la velocidad como las acciones detectadas por la solución se almacenan en formato csv, en archivos independientes. El nombre de los ficheros empieza por *state* o *vel* seguido del nombre del archivo del video estudiado y en cada uno la información viene identificada por la escena a la que corresponde.

En lo concerniente a la representación de resultados, al ser una tarea post ejecución, se ha creado un script en Python externo al sistema el cual extrae la velocidad del fichero y la representa en función de la escena. Dicho script además extrae la información de los estados detectados y, con los videos previamente clasificados en formato csv, interpola ambas fuentes creando una matriz de confusión a modo de resultado de verosimilitud de la solución propuesta.

La información de interés se puede dividir en dos conjuntos, aquella información resultado de la ejecución del algoritmo de visión artificial y sobre el estado/salud de la plataforma.

- Información del número de escena y personas detectadas con sus coordenadas asociadas(x,y,z).
- Velocidades asociadas a cada persona detectada por escena.
- Archivo de *log* del aplicativo completo.
- Información del estado del disco en periodos de 10 minutos.
- Información del estado de la memoria RAM por minuto.
- Información del estado del consumo de CPU por minuto.

Con el fin de separar la parte de servicio de la parte de gestión, aunque la máquina destino es la misma, para exportar la información se pueden utilizar rutas destino distintos. Los principales motivos para esto es poder separar ambos entornos en distintos volúmenes, facilitar la tarea de depuración de errores y extraer la información resultante.

3.7. Cuenta de gitlab instalada

Con el fin de mejorar la accesibilidad al código fuente de la plataforma, la actualización por parte de los desarrolladores y la portabilidad de todo el *software* de la forma más simple sin tener que recurrir a redes virtuales privadas o similares, se ha configurado un repositorio privado alojado en GitLab. En él se encuentra toda la estructura de Ansible necesaria para el *despliegue* de la plataforma y mantenimiento de esta.

Capítulo 4.

Resultados

En este capítulo se muestran los resultados obtenidos durante la realización de este TFG. Entre los puntos a tratar como resultados inherentes del desarrollo del proyecto, son el detector de personas 3.2 , la asociación y seguimiento de personas 3.3 y detección de acciones 3.4.

Todos los videos utilizados en los experimentos tienen unas características comunes:

- Imágenes capturadas desde una posición cenital con una cámara Kinetc v2.
- Una persona por escena y realizando una acción por video, entre las que se encuentran:
    - Andando.
    - Corriendo.
    - Parado.
    - Cayendo.
- Un entorno estático, sin movimiento de objetos.

4.1. Análisis de resultados de la detección de personas.

En este apartado se realiza una comparación de los resultados obtenidos mediante el algoritmo implementado para la detección de personas.

Para la evaluación se han utilizado imágenes de profundidad, videos en el que aparecen distintas personas de altura variable y secuencias en las que se realizan distintas acciones.

- Secuencia 1. Un único sujeto andando en la escena Figura 4.1.
- Secuencia 2. Un único sujeto corriendo en la escena Figura 4.2.
- Secuencia 3. Un único sujeto parado en la escena Figura 4.3.
- Secuencia 4. Un único sujeto cayendo en la escena Figura 4.4.

4.1.1. Análisis de cualitativo.

Se puede observar en las secuencias anteriormente mencionadas, que tras un tiempo en escena se detecta completamente a la persona presente en ella. También se aprecia el ruido en los márgenes de la imagen, provocando que cuando se acerca una persona a estos, una parte mínima del ruido se integra en la detección. La Figura 4.4 tiene lugar la caída de una persona, y en este caso se detecta a la persona en movimiento cayendo y cuando se levanta, pudiendo seguir la secuencia del movimiento.

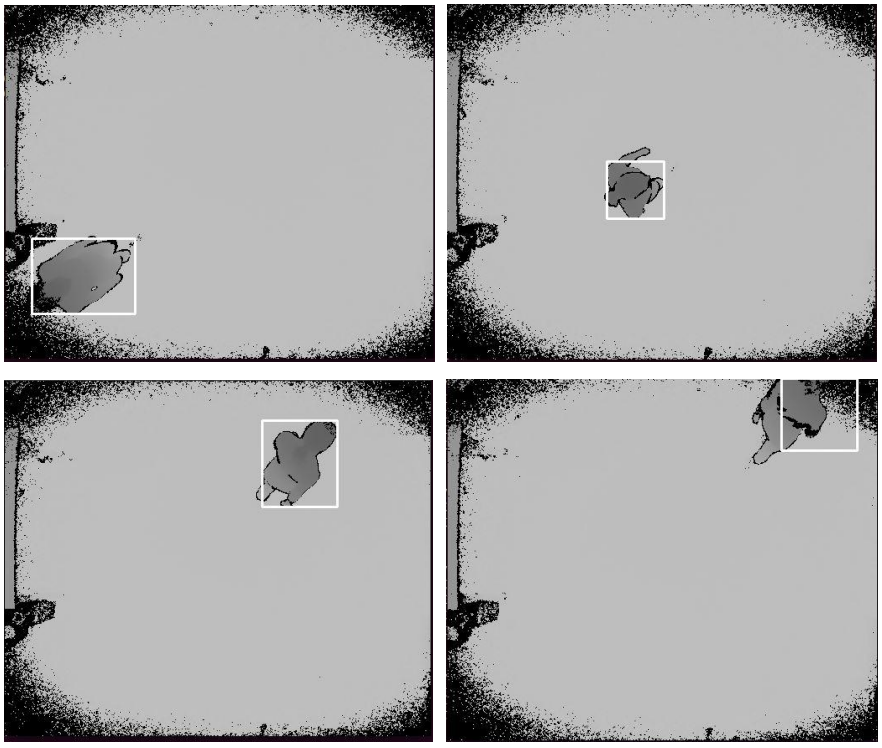

Figura 4.1 Detección de un único sujeto andando en la escena.

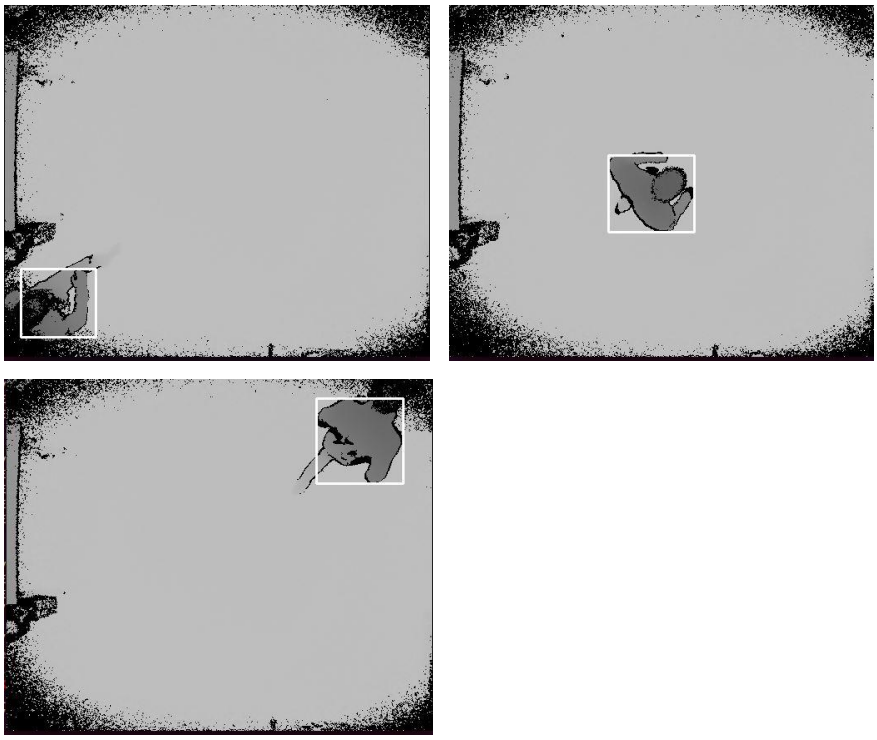

Figura 4.2 Detección de un único sujeto corriendo en la escena.

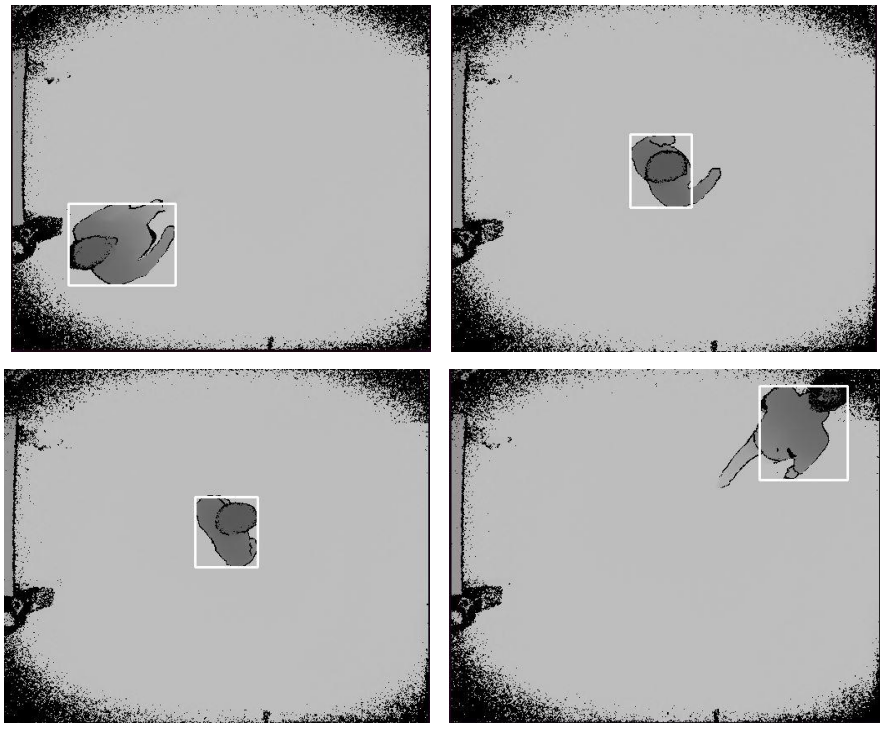

Figura 4.3 Detección de un único sujeto parado en la escena.

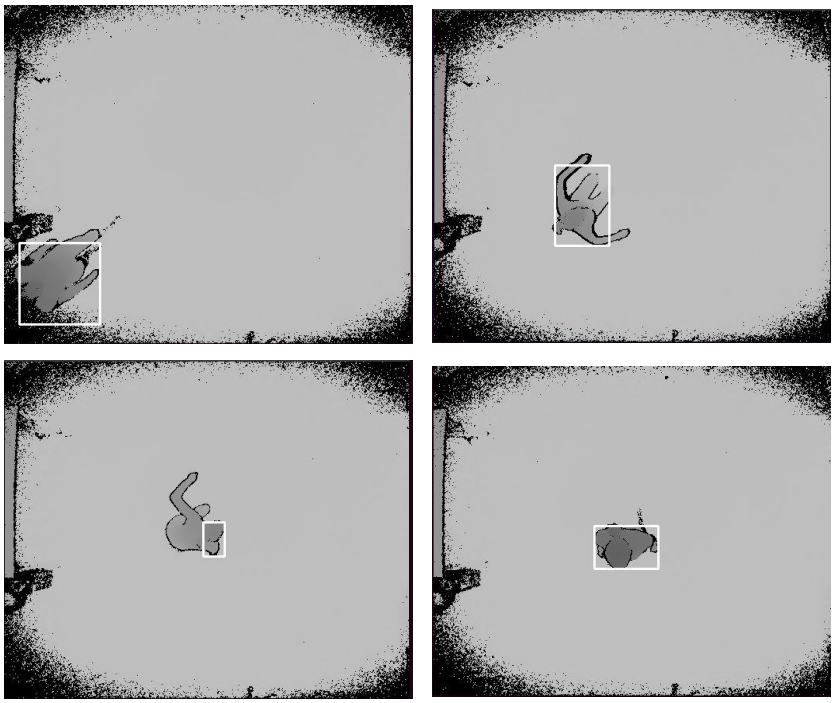

Figura 4.4 Detección de un único sujeto cayendo en la escena.

4.1.2. Análisis cuantitativo.

Para evaluar del algoritmo se ha utilizado métricas de calidad, una técnica ampliamente utilizada cuando la respuesta del sistema es un valor binario. Dichas métricas son:

- Verdaderos positivos (VP): Número de elementos etiquetados como personas y que nuestro algoritmo detecta como personas.
- Falsos positivos (FP): Número de elementos etiquetados como no personas y que nuestro algoritmo detecta como personas.
- Verdaderos negativos (VN): Número de elementos etiquetados como no personas y que nuestro algoritmo detecta como no personas.
- Falsos negativos (FN): Número de elementos etiquetados como personas y que nuestro algoritmo detecta como no personas.
- Precisión: Verdaderos positivos en realmente son positivos en comparación con el número total de valores positivos predichos.
- Sensibilidad (*Recall)*: Verdaderos positivos que el modelo ha clasificado en función del total de valores positivos.

En la Tabla 4.1 se muestran los valores métricos calculados expresados en porcentaje, sobre el número total de imágenes con una persona etiquetada para los valores VP y FP, y sobre el número total de imágenes sin ninguna persona etiquetada para los valores VN y FN.

Para la generación de estos datos, se han excluido los datos utilizados en el entrenamiento del algoritmo.

		Resultado de la detección	
		Positivo	Negativo
Valor actual	Positivo	97,552%	2,448%
	Negativo	10,450%	89,550%

Tabla 4.1. Resultados cuantitativos de la detección de personas.

$$Precision = \frac{VP}{VP + FP} = 90{,}324\% \qquad 4.1$$

$$Sensibilidad\ (Recall) = \frac{VP}{VP + FN} = 97{,}552\% \qquad 4.2$$

A pesar de que el valor de precisión obtenido no es muy alto, ecuación 4.1 , se considera suficiente dado que la detección no es el objetivo principal de este trabajo. Se ha comprobado que muchos errores se producen cuando la persona se encuentra cerca del borde de la escena, donde existe una gran cantidad de ruido dificultando la detección.

4.2. Análisis de resultados del seguimiento de personas.

En este apartado se realiza una comparación de los resultados obtenidos mediante el algoritmo implementado para el seguimiento de personas y resultados obtenidos con el mismo. Para el entrenamiento se han utilizado imágenes de profundidad, videos en el que aparecen distintas personas de altura variable y secuencias en las que se realizan distintas acciones. Para la generación de estos datos, se han excluido los datos utilizados en el entrenamiento del algoritmo.

- Secuencia 1. Un único sujeto andando en la escena Figura 4.5.
- Secuencia 2. Un único sujeto corriendo en la escena Figura 4.6.
- Secuencia 3. Un único sujeto parado en la escena Figura 4.7.
- Secuencia 4. Un único sujeto cayendo en la escena Figura 4.8.

4.2.1. Análisis cualitativo.

Se puede observar en las secuencias anteriormente mencionadas, que tras un tiempo en escena se detecta la acción con un alto grado de verosimilitud. Como ocurre en la detección de personas se aprecia el ruido en los márgenes de la imagen Figura 4.5, provocando que cuando se acerca una persona a estos, la acción no se pueda concretar de forma precisa. En la Figura 4.8 tiene lugar la caída de una persona y la correcta detección de la acción.

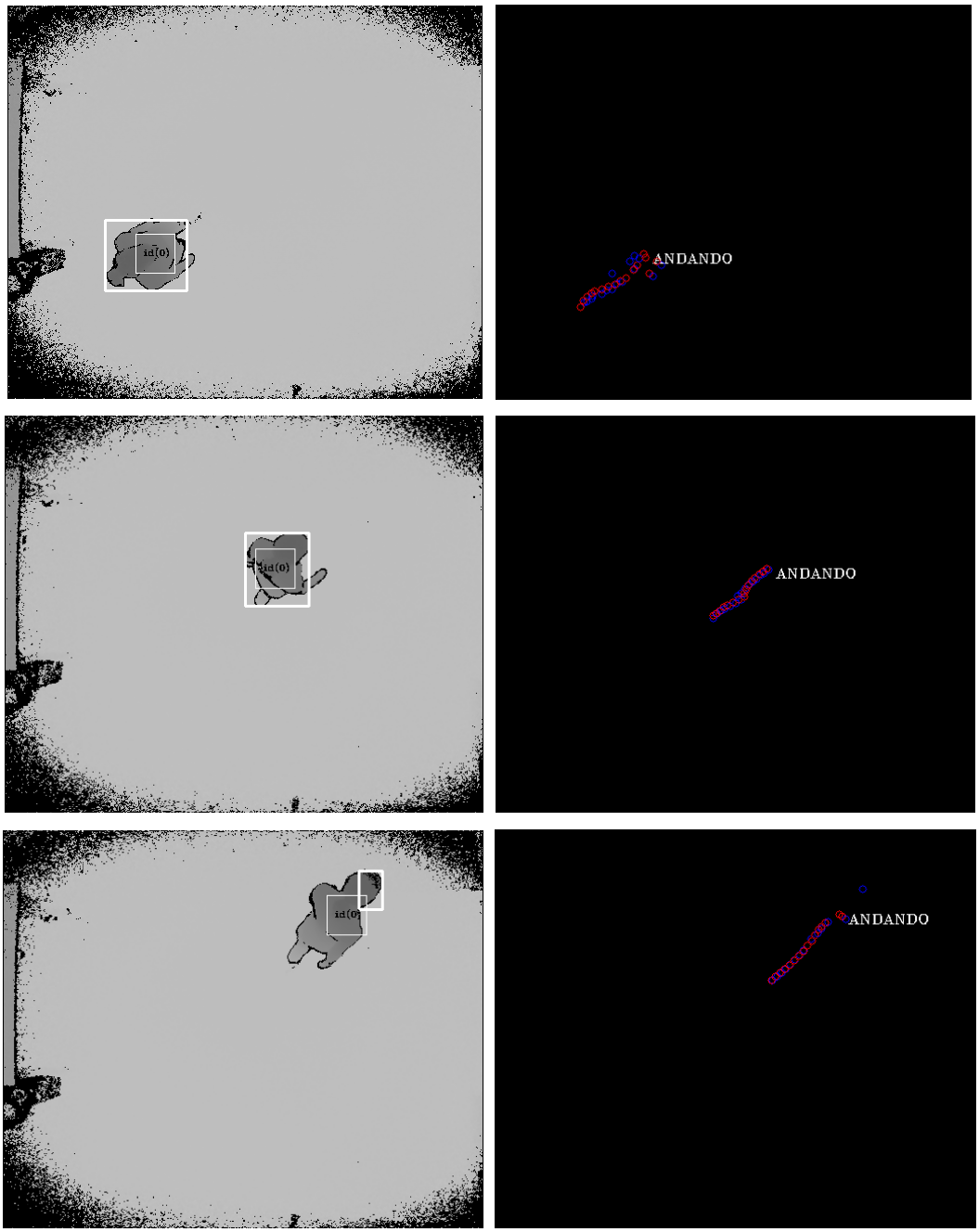

Figura 4.5 Seguimiento de un único sujeto andando en la escena.

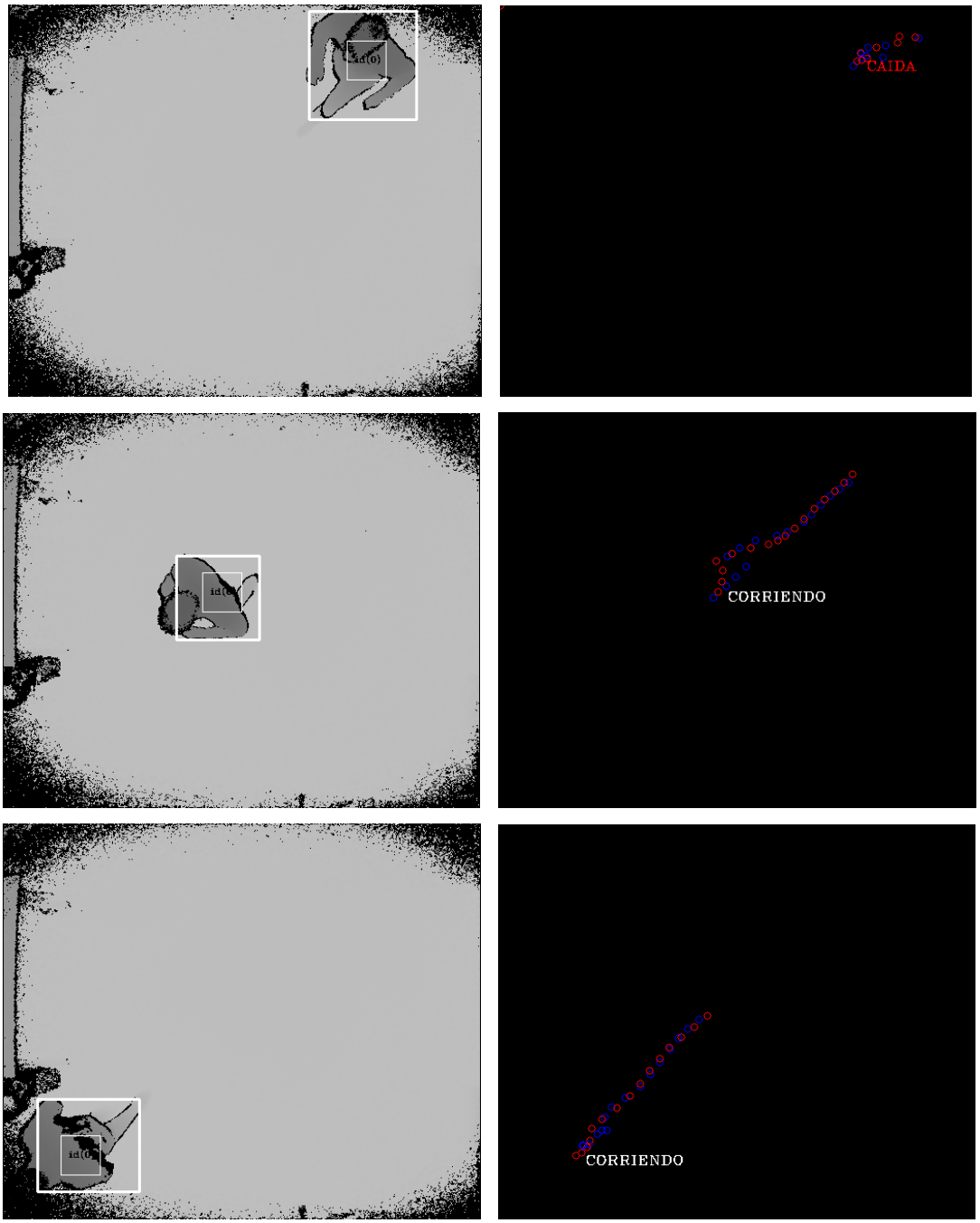

Figura 4.6 Seguimiento de un único sujeto andando en la escena.

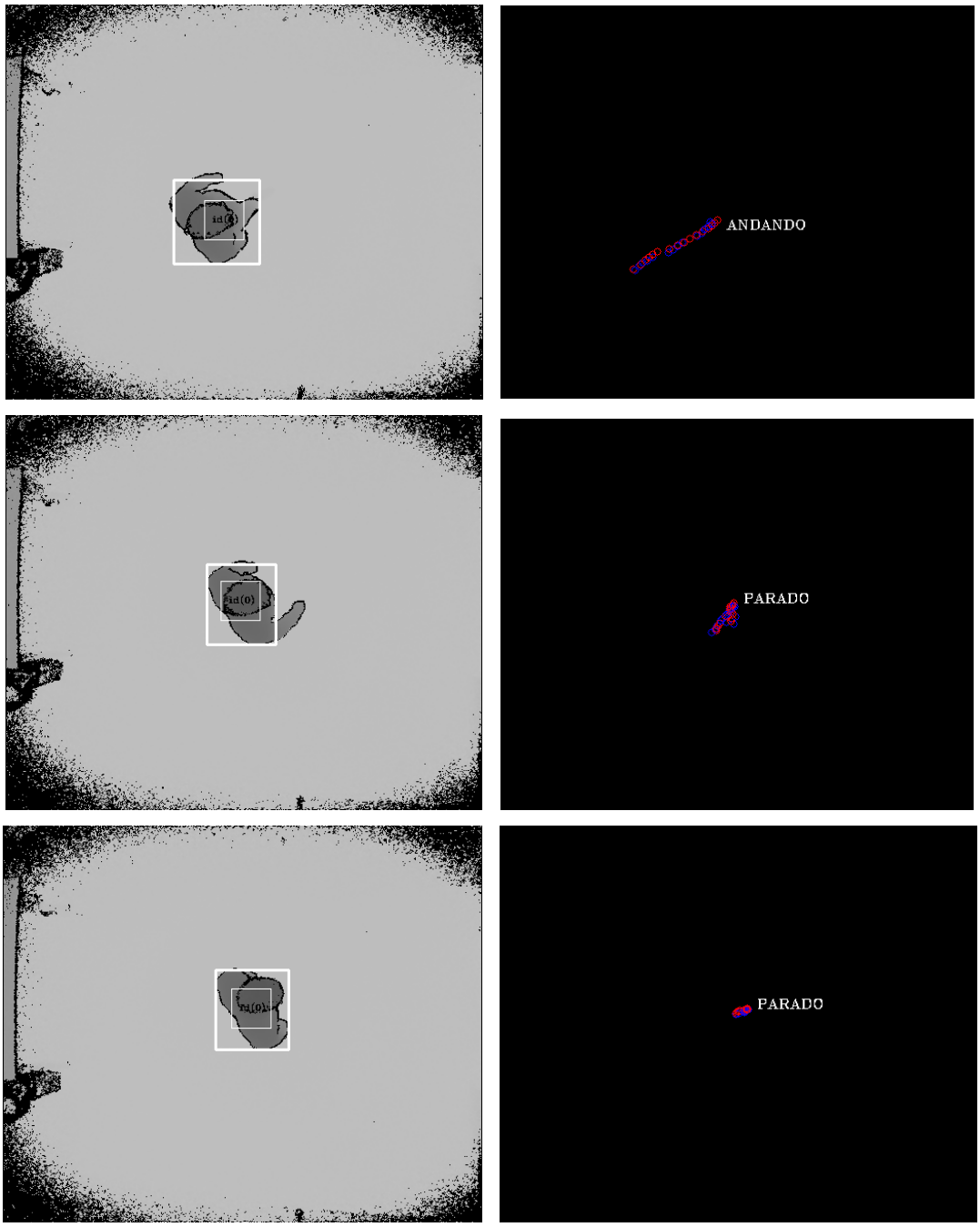

Figura 4.7 Seguimiento de un único sujeto parado en la escena.

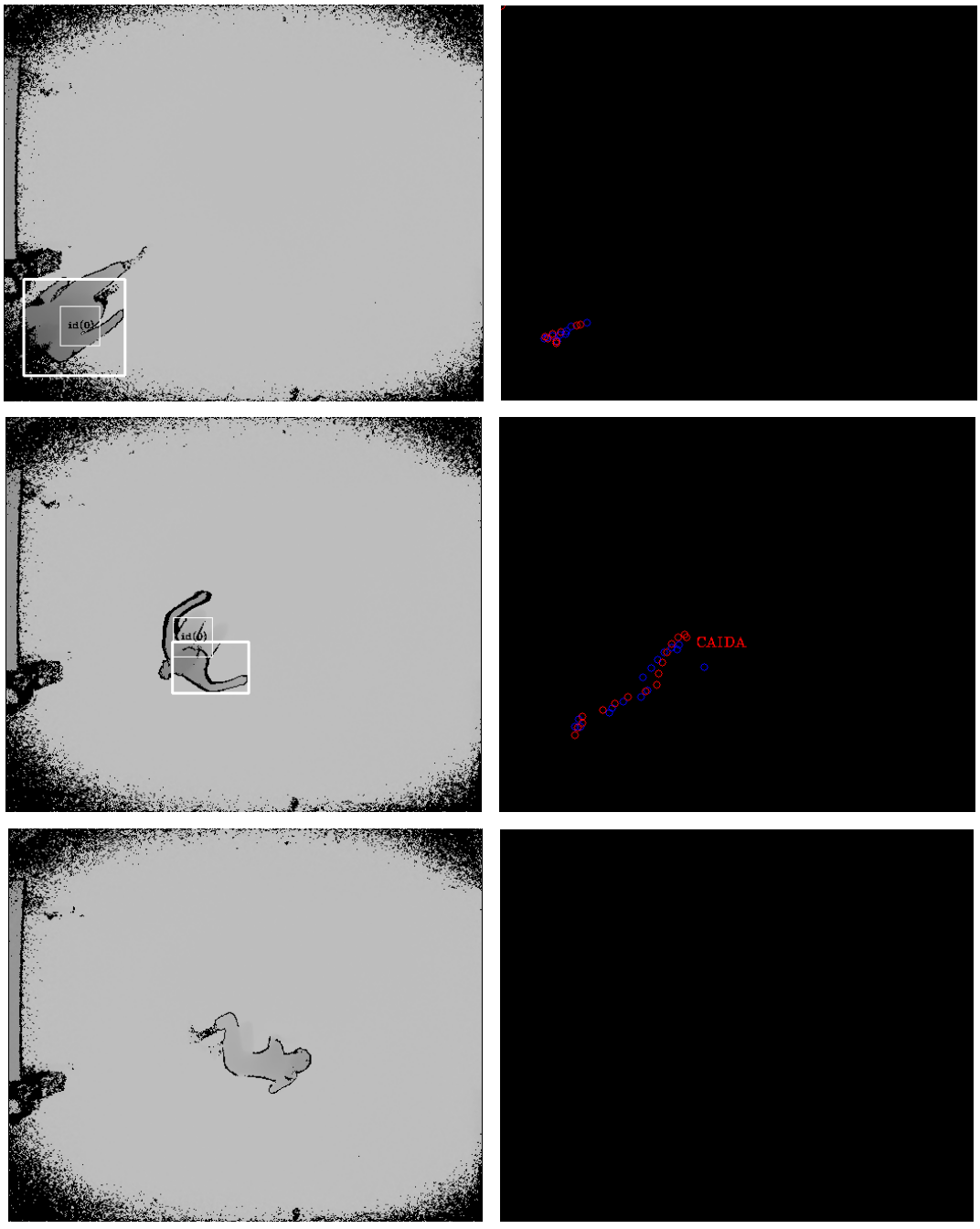

Figura 4.8 Seguimiento de un único sujeto cayendo en la escena.

4.2.2. Análisis cuantitativo de resultados.

Para evaluar los resultados arrojados por el algoritmo implementado se ha utilizado una matriz de confusión, ampliamente implementadas cuando se trabaja en el campo de la inteligencia artificial y en especial en el problema de la clasificación estadística [29]. Una matriz de confusión es una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado.

Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo dos clases. Si en los datos de entrada el número de muestras de clases diferentes cambia mucho la tasa de error del clasificador no es representativa de lo bien que realiza la tarea el clasificador.

En la siguiente Tabla 4.2 se muestran los valores métricos calculados expresados en porcentaje sobre el número total de videos de cada acción, entendiendo que por cada video aparece una persona. Se observa que existe una precisión menor entre los valores de andar y parar, esto se debe a que la acción de parar es una acción compuesta y por tanto hay cierto tiempo de adaptación para detectar el cambio entre ambas escenas. De una forma parecida ocurre con andar y caer, pero en menor medida.

		Acción Calculada			
Acción Real	Frames	Correr	Andar	Parar	Caer
Correr	171	95,322%	2,924%	0,000%	1,754%
Andar	1590	4,717%	80,126%	13,899%	1,258%
Parar	188	0,000%	13,830%	86,170%	0,000%
Caer	27	0,000%	7,407%	0,000%	92,593%

Tabla 4.2 Resultados cuantitativos de la detección de acciones.

4.3. Evaluación de tiempo de ejecución

Se ha estudiado el tiempo de ejecución total del algoritmo implementado por escena para estudiar la viabilidad con una integración en un sistema con imágenes en tiempo real. Se ha centrado el estudio, en el tiempo que transcurre al recibir la información de una escena en la que hay una persona hasta la detección de la acción. El tiempo resultante es la media de varias ejecuciones con videos de testeo aleatorios.

$$T_{medio} = 0{,}280\, ms \qquad 4.3$$

Como se puede observar, se podría integrar sin problemas con sistemas de video digital con una velocidad o *framerate* de 60 imágenes por segundo. También los proporcionados por la Kinect, cámara utilizada en este trabajo, la cual graba con una velocidad de 30 fotogramas al segundo.

Capítulo 5.

Conclusiones y líneas futuras

5.1. Conclusiones y líneas futuras

Este trabajo se ha centrado en el seguimiento y detección de acciones mediante la utilización de sensores ToF en posición cenital.

En primer lugar, se ha analizado e implementado un detector de personas a partir de imágenes de profundidad. En las imágenes solo aparece una única persona sobre un fondo estático realizando una acción, entre estas acciones se encuentra correr, andar, parar y caer. Para detectar a la persona esta debe tener unas dimensiones mínimas sin importar su altura, además sobre la imagen se realiza una resta de fondo para eliminar los objetos estáticos de la escena.

Para el seguimiento de las personas, se ha incorporado un banco de filtros de Kalman, que permite el seguimiento robusto de un número variable de personas, así como la estimación de su posición y velocidad en tres dimensiones. Por último, se estudia la acción que están realizando todas las personas que se consideran presentes en la escena para sucesivas iteraciones, aunque solo se consideren válidas dichas acciones calculadas tras un periodo de tiempo. El cálculo de la acción que se está llevando a cabo se realiza en base a unos parámetros heurísticos calculados con la información de entrenamiento.

Para la validación experimental del sistema implementado se han utilizado dos métodos distintos de análisis, métricas de calidad y matriz de confusión respectivamente. Se concluye que se detecta de forma correcta con una 98% de sensibilidad y una precisión del 90% a las

personas presentes en la escena, y en torno al 89% de las acciones reales, que han permitido validar el sistema.

Toda la solución propuesta se ha diseñado siguiendo los principios SOLID con el fin de proveer al sistema de modularidad y robustez, además de una mejor integración con otros sistemas y componentes.

5.2. Líneas de trabajo futuro

Al finalizar este trabajo se han contemplado distintas vías por las cuales proseguir y mejorar distintos aspectos:

- **Mejoras en la detección de personas**: Implementar un sistema con una cámara ToF capaz de obtener unas prestaciones parecidas en exteriores que las prestaciones que ofrece actualmente la Kinect v2 en interiores. Además, estudiar la implementación de un detector de personas de análisis de componentes principales (PCA).
- **Estudio de la detección de acciones cambiando las condiciones del entorno**: En este trabajo se ha utilizado videos en los que aparece una persona en un entorno estático. En los futuros estudios se puede sustituir tanto el número de personas en escena como el tipo de entorno.
- **Estudio de la detección de acciones con imágenes capturadas desde un plano picado**: En este trabajo todas las imágenes utilizadas han sido capturadas desde un plano cenital, las imágenes desde un plano picado aportan más información y con esto se podría mejorar la verosimilitud de las acciones medidas hasta ahora e incluir otras nuevas.

Bibliografía

[1] Geintra, «Página web del grupo de investigación geintra,» 25 Abril 2017. [En línea]. Available: http://www.geintra-uah.org/.

[2] D. Fuentes Jímenez, Diseño, implementación y evaluación de un sistema de conteo de personas basado en cámaras de tiempo de vuelo., Alcalá de Henares: Escuela Pólitecnica Superior, Universidad de Alcalá, 2016.

[3] M. Higuera Pinillos, «Universidad de Alcalá de Henares - TFG,» 2018. [En línea]. Available: https://ebuah.uah.es/dspace/bitstream/handle/10017/33821/TFG_%20Higuera_Pinillos_2018.pdf;jsessionid=589731392EB9B981722C49CFF0B3F6F7?sequence=1. [Último acceso: 18 05 2019].

[4] J. P.O'Connor, «The Xbox one system on a chip and Kinect sensor,» *Micro,IEEE,* vol. 34, nº 2, pp. 44-53, 2014.

[5] J. Rubira, «GenBeta,» 14 Julio 2011. [En línea]. Available: https://www.genbeta.com/desarrollo/solid-cinco-principios-basicos-de-diseno-de-clases. [Último acceso: 19 02 2019].

[6] P. Gil, T. Kislerb, G. García, C. Jara y J. J. Corrales, «Calibración de cámaras de tiempo de vuelo: Ajuste adaptativo del tiempo de integración y análisis de la frecuencia de modulación,» *Revista Iberoamericana de Automática e Informática industrial,* pp. 453-464, 2013.

[7] B. Alcántara, «Andro4All,» Andro4All, 18 08 2019. [En línea]. Available: https://andro4all.com/2019/08/camaras-tof-que-son-moviles. [Último acceso: 03 05 2020].

[8] G. I. Solutions, «GMV Blog,» GMV, [En línea]. Available: https://www.gmv.com/es/Sectores/Espacio/segmento_espacial/autonomia_robotica.html. [Último acceso: 03 05 2020].

[9] E. Rufo Merino, Calibración de array cámaras ToF para reconstrucción volumétrica, Alcala de Henares: UAH, 2013.

[10] M. Frank, M. Plaue, H. Rapp, U. Köthe, B. Jähne y F. Hamprecht, «Theoretical and experimental error analysis of continous-wave time-offlight range cameras.,» de *Optical Engineering 48*, 2009, pp. 13602-13618.

[11] Grupo de Inteligencia Artificial Aplicada de la, UC3M, «Portal Universidad Carlos III,» 15 10 2010. [En línea]. Available: http://portal.uc3m.es/portal/page/portal/actualidad_cientifica/noticias/camara_videojuego/15_10_2010_el_potencial_de_las_camaras_3.pdf. [Último acceso: 2018 08 17].

[12] S. Fuchs, Multipath interference compensation in time-of-flight camera images, 20th International Conference on Pattern Recognition: IEEE, 2010.

[13] S. Lee, B. Kang, J. D.K. Kim y C. Yeong Kim, Motion blur-free time-of-flight range sensor, IS&T/SPIE Electronic Imaging, 2012.

[14] D. Jiménez, D. Pizarro, M. Mazo y S. Palazuelos, «Single frame correction of motion artifacts in PMD-based time of flight cameras,» *IEEE Conference on Computer Vision and Pattern Recognition,* p. 893–900, 2012.

[15] D. Jimenez, D. Pizarro y M. Mazo, «Single frame correction of motion artifacts in PMD-based time of flight cameras,» *Image Vis. Comput.,* vol. 32, nº 12, pp. 1127-1143, 2014.

[16] R. E. Kalman, «New Approach to Linear Filtering and Prediction,» *Journal of Basic Engineering,* nº Marzo, pp. 35-45, 1960.

[17] G. W. Bishop, «An Introduction to the Kalman Filter,» Department of Computer Science University of North Carolina at Chapel Hill, North Carolina, USA, 2006.

[18] O. L. .. R. Jacobs, «Introduction to Control Theory, 2nd Edition,» Oxford University Press, London, GB, 1993.

[19] R. B. Hwang y P.Y.C., Introduction to Random Signals and Applied Kalman Filtering, Second Edition, John Wiley and Sons, Inc., 1992.

[20] A. Mohinder S. Grewal y P. Angus, Kalman Filtering Theory and Practice, Upper Saddle River, New Jersey, USA: Prentice Hall, 1993.

[21] M. A. Mendoza Perez, Reconocimiento de acciones humanas basado en modelos probabilisticos de espacio de estados., Granada: Universidad de Granada, 2009.

[22] A. F. Bobick y Y. A. Ivanov, «Action recognition using probabilistic parsing.,» *Proceedings of The IEEE - Conference on Computer Vision and Pattern Recognition,* p. 196–202, 1998.

[23] M. Pittore, C. Basso y A. Verri, «Representing andrecognizing visual dynamic events with support vec-tor machines.,» *Proceedings of the International Conference on Image Analysis and Processing,* pp. 18-23, 1999.

[24] Y. Guo, G. Xu y S. Tsuji, «Understanding human motion patterns.,» *Proceedings of The International Conference on Pattern Recognition,* vol. II, pp. 325-329, 1994.

[25] K.-H. Jo, Y. Kuno y Y. Shirai, «Manipulative handgesture recognition using task knowledge for human computer interaction.,» *Proceedings of the IEEE In-ternational Conference on Automatic Face and Gesture Recognition,* vol. I, p. 468–473, 1998.

[26] StackShare, "StackShare," 12 06 2020. [Online]. Available: https://stackshare.io/stackups/docker-vs-vagrant-vs-red-hat-openshift#stats. [Accessed 12 06 2020].

[27] D. Inc., «Docker Official Web,» Docker Inc., 29 10 2019. [En línea]. Available: https://www.docker.com/.

[28] RedHat, «Viewing and Managing Log Files,» de *https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/ch-viewing_and_managing_log_files*.

[29] Wikipedia, «Wikipedia,» Fundación Wikimedia, Inc, 12 07 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Matriz_de_confusi%C3%B3n. [Último acceso: 03 08 2020].

[30] Eglitis-media, «Datosmundial.com,» 2016. [En línea]. Available: https://www.datosmundial.com/estatura-promedio.php. [Último acceso: 23 08 2019].

Apéndice A Manual de usuario

Este manual de usuario presenta el uso del software desarrollado en este proyecto. Las funciones aquí implementadas, han sido desarrolladas en C/C++ sobre las bibliotecas *OpenCV 2.4.6.1* y virtualizado con *Docker Engine v19.03.12*. Esto hace necesario únicamente una instalación previa de *Docker Engine v19.03.12* (o superior) en un sistema operativo con entorno gráfico.

A.1. Preparación del entorno

Tras tener instalados los componentes asociados a Docker, se va a realizar la preparación del sistema operativo anfitrión para desplegar el container. Lo pasos que seguir son:

1. Creación de los volúmenes lógicos: Bajo la premisa que un container es un servicio y por tanto tras terminar este el container debe desaparecer. Se han creado unos volúmenes lógicos para almacenar los ficheros de resultados y la información de la ejecución para poder acceder a ellos, y asi recolectar la información tras el procesado del video.

```
docker volume create statics
docker volume create logs
```

2. Recursos gráficos: Al realizar representación de imágenes, es necesario utilizar los recursos gráficos del sistema operativo, algo que de forma nativa no es compartido por el sistema operativo y el container.

```
xhost +
```

3. Inicialización del container: Tras realizar los pasos anteriores el sistema operativo se encuentra preparado para inicializar el servicio. Los *flags* o argumentos necesarios para el buen funcionamiento significan:
    a. *–rm* –> Tras la finalización del programa el container se eliminará.
    b. *–v $(path_video):/home/x/Thea/200_SOURCE_v2/video* –> Se añade la ruta origen de la máquina (*path_video)* donde se encuentran los videos para procesar al container.
    c. *–v logs:/home/x/Thea/200_SOURCE_v2/logs/* –> Se añade el volumen lógico creado en el paso 1, para poder consultar la información de los *logs* tras la finalización del servicio.
    d. *–v statics:/home/x/Thea/200_SOURCE_v2/statics/* –> Se añade el volumen lógico creado en el paso 1, para poder consultar la información de las estadísticas y estado tras la finalización del servicio. Se incluye la información de velocidad y los estados detectados.
    e. *--net=host* –> La red se configura en el container para compartir el mismo segmento que la máquina anfitriona, y poder utilizar los recursos gráficos.
    f. *--ipc=host* –> El contexto del programa configura en el container para compartir el mismo que la máquina anfitriona, y poder utilizar las señales propias del sistema operativo anfitrión.
    g. *-v /tmp/.X11-unix:/tmp/.X11-unix* –> Se añade el *path* necesario para utilizar los recursos gráficos del sistema operativo.
    h. *-e DISPLAY=$DISPLAY* –> Se añade la variable de entorno del recurso gráfico gestionado por el sistema operativo.
    i. *-it selene:v13* –> Imagen base sobre la que se crea el container.
    j. *./Selene act-CLG-000* –> Comando a ejecutar como servicio principal por el container. El último elemento se refiere al video a procesar, el sistema admite hasta 28 videos los cuales se van a procesar de forma secuencial.

```
docker run --rm -v $(path_video):/home/x/Thea/200_SOURCE_v2/video -v logs:/home/x/Thea/200_SOURCE_v2/logs/ -v statics:/home/x/Thea/200_SOURCE_v2/statics/ --net=host --ipc=host -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=$DISPLAY -it selene:v13 ./Selene act-CLG-000
```

A.2. Almacenamiento de resultados

Los resultados se almacenan en los volúmenes mencionados anteriormente, tras la ejecución. En el volumen de *logs* se encuentra la información de las subrutinas, con la información de las clases invocadas, métodos utilizados y flujos de estos. Por cada clase principal se crea un fichero unitario, para poder realizar un seguimiento más pormenorizado del programa. Todos los archivos tienen la misma estructura, nivel de criticidad, directiva ejecutada y la información representada.

```
root@ubuntu:/# ls /var/lib/docker/volumes/logs/_data
actions_speed
Kalman_procces
StatusVector_Control
utilities_image
root@ubuntu:/# tail -n 10  /var/lib/docker/volumes/logs/_data/StatusVector_Control
| Info | StatusVectorControl::check_coordinate | Final execute rutine StatusVectorControl::check_coordinate
| Info | StatusVectorControl::present_candidate | Init execute rutine StatusVectorControl::present_candidate
| Debug | StatusVectorControl::present_candidate | Value to Coordinate[0][4][0] = 50
| Debug | StatusVectorControl::present_candidate | Value to Coordinate[0][4][1] = 369
| Debug | StatusVectorControl::present_candidate | Value to Coordinate[0][4][2] = 122
| Debug | StatusVectorControl::present_candidate | Value to VectorEstado[0] = 5
| Info | StatusVectorControl::present_candidate | Final execute rutine StatusVectorControl::present_candidate
```

En el otro volumen creado, se almacena la información relacionada con el reconocimiento de personas y reconocimiento de acciones, el formato de estos archivos es CSV (*Comma Separated Values*). De cada video procesado se crean tres archivos:

- Información del detector de personas:
    - Se almacena el número de escena y si se ha detectado una persona o no.
    - Ruta: $(path_volumen_statics)/detector/det_$(nombre_video)

- Información básica de la detección de acciones:
    - Se almacena el número de escena y la acción detectada.
    - Ruta: $(path_volumen_statics)/state/state_$(nombre_video)
- Información extendida de la detección de acciones:
    - Se almacena el número de escena, número de usuario, acción detectada, número de medidas recolectadas desde que el usuario ha sido detectado, media de las velocidades desde que el usuario ha sido detectado, velocidad en el momento actual y varianza de los datos.
    - Ruta: $(path_volumen_statics)/state/state_ext_$(nombre_video)

```
root@ubuntu:/# tail -n 4 /var/lib/docker/volumes/statics/_data/detector/det_act-CLG-001.csv
182;0
183;0
184;0
185;0
root@ubuntu:/# tail -n 4 /var/lib/docker/volumes/statics/_data/state/state_act-CLG-001.csv
153;1
154;1
155;1
156;1
root@ubuntu:/# tail -n 4 /var/lib/docker/volumes/statics/_data/state/state_ext_act-CLG-001.csv
153;0;1;15;140;17;135.620795
154;0;1;16;145;17;139.889242
155;0;1;17;147;15;141.237389
156;0;1;18;146;15;139.710415
```

Apéndice B Pliego de condiciones

B.1. Pliego de Condiciones

Para el buen funcionamiento del trabajo completo serán necesario provisionar sistema con el hardware y el software necesario:

B.2. Requisitos de Hardware: (para una única máquina)

- 16 Gb de Ram DDR3 a
- Procesador de 64 bits i7
- Al menos 100 Gb de memoria en disco, para las librerías requeridas, el software desarrollado.

B.3. Requisitos de Software:

- Sistema operativo Ubuntu 20.04 LTS
- Librerias OpenCv 2.4.9
- Docker Engine v19.03.12
- Git 1.9.1
- Cmake 2.8.12.2

Apéndice C Presupuesto

C.1. Costes de equipamiento:

- Costes de Hardware:

Concepto	Cantidad	Coste unitario	Subtotal (Euros)
Ordenador Medion Erazer i7 16Gb	1	1300 €	1300 €
Cámara ToF Kinect v2	1	169 €	169 €
Subtotal Hardware (Euros)			**1469 €**

- Costes de Software:

Concepto	Cantidad	Coste unitario	Subtotal (Euros)
Sistema operativo Ubuntu 20.04 LTS	1	0 €	0 €
Librerias OpenCv 2.4.9	1	0 €	0 €
Docker Engine v19.03.12	1	0€	0€
Rsyslog 7.4.4	1	0 €	0 €
Git (GitLab) 1.9.1	1	0 €	0 €
Word	1	90 €	90 €
Cmake 2.8.12.2	1	0 €	0 €
Subtotal Software (Euros)			**90 €**

C.2. Costes mano de obra:

Concepto	Cantidad	Coste unitario	Subtotal (Euros)
Desarrollo software	180	67 € /hora	12.060 €
Documentación	100	15 € /hora	1500 €
Coste total mano de obra			**13.560 €**

El coste total del proyecto asciende a 13.650 €.

Apéndice D

Sintaxis de las clases

Archivos: Draw.cpp

Draw.hpp

Clases y métodos:

```
class drawImage{
    private:
        uint32_t readimage;
        uint16_t Imagen[434176];
        char id_aux[16],Vel_asoc[20];
        uint8_t user_detected;
        uint8_t velocidad,aux;
        uint16_t*** Coordinate;
        uint16_t*** Kalman_medidas;
        uint16_t* VectorEstado;
        int coordinate_user;
        int minValv,maxValv;
        uint8_t rectangulo_desf;
        uint32_t cont_seq;
        int alturamaxima,alturaminima;

    public:
        drawImage();
        void set_Coordinate_pointer (uint16_t*** pointer_Coordinate);
        void            set_Kalman_medidas_pointer            (uint16_t***
pointer_Kalman_medidas);
        void set_VectorEstado_pointer (uint16_t* pointer_VectorEstado);
        void set_coordinate_user( int coordinate_user_value );
        void set_cont_seq( uint32_t seq );
        //void drawImage ( cv::Mat & image );  //Deprecated
```

```
        void drawKalman( cv::Mat & image );
        void drawInfo(cv::Mat & image );
        void drawClasificador(cv::Mat & image);
        void drawConteo(cv::Mat & image);
        void draw();
};
```

Archivos: Utilities.cpp

Utilities.hpp

Clases y métodos:

```
class image{
    private:
        int alturamaxima,alturaminima,coordenada_aux;
        uint32_t num_frame;
        uint16_t margenciego_x,margenciego_y;
        uint16_t alturamax_frame,alturamax_frame_user,counter_candidates;
        uint32_t altura_fram_all;
        uint8_t cont_user;
        uint16_t* buffer_minimos;
        int coordenada_EMA;
    public:
        image();
        int count_person(uint16_t bufferimagen[434176]);
        void set_alturamaxima(int altura_maxima);
        void set_alturaminima(int altura_minima);
        void set_num_frame(int frame);
        void     set_margenciego(uint16_t     margenciego_x,     uint16_t
margenciego_y);
        void set_no_user ();
        int get_count_person(){ return cont_user; }
        int get_num_frame(){ return num_frame; }
        int get_coordinate(){ return coordenada_aux; }
        uint16_t get_alturaframe(){ return alturamax_frame; }
        void set_buffer_minimos_pointer (uint16_t* pointer_buffer_minimos);
};
```

Archivos: Kalman_process.cpp

Kalman_process.hpp

Clases y métodos:

```
   class KalmanActions{
       private:
           uint32_t cont_seq;
           unsigned int type;
           cv::Mat state;      //[x,      y,     z,   v_x,   v_y,    v_z]
           cv::Mat meas;       // [z_x,    z_y,   z_z]
           int stateSize;
           int measSize;
           int contrSize;
           double precTick,dT,ticks;
           uint16_t*** Coordinate;
           uint16_t*** Kalman_medidas;
           uint16_t* VectorEstado;
           uint16_t VectorEstadoKalman;
           logger_txt logger_kalman_proc;
           char log[100],file_logger[100];
           cv::KalmanFilter    new_object_meas(uint8_t    id,cv::KalmanFilter
kalman_filter);
           cv::KalmanFilter    old_object_meas(uint8_t    id,cv::KalmanFilter
kalman_filter);

       public:
           KalmanActions();
           void set_Coordinate_pointer (uint16_t*** pointer_Coordinate);
           void            set_Kalman_medidas_pointer            (uint16_t***
pointer_Kalman_medidas);
           void set_VectorEstado_pointer (uint16_t* pointer_VectorEstado);
           void set_VectorEstadoKalman (uint16_t Vector_EstadoKalman);
           void set_cont_seq( uint32_t seq);
           void set_updateContSeq();
           uint16_t get_VectorEstadoKalman(){ return VectorEstadoKalman;}
           cv::KalmanFilter       Configure(uint8_t       id,cv::KalmanFilter
kalman_filter);
           cv::KalmanFilter       Predictor(uint8_t       id,cv::KalmanFilter
kalman_filter);
           cv::KalmanFilter       Corrector(uint8_t       id,cv::KalmanFilter
kalman_filter);
```

```
};
```

Archivos: StatusVectorControl.cpp

StatusVectorControl.hpp

Clases y métodos:

```
class StatusVectorControl{
    private:
        uint8_t candidate,Vector_dif[3],Vector_dif_aux[3];
        uint8_t umbral;
        uint32_t cont_seq;
      uint8_t new_userID;
        uint16_t* VectorEstado;
        uint16_t** Coordinate_aux_vect;
        uint16_t*** Coordinate;
      uint16_t*** Kalman_medidas;
        uint16_t VectorEstadoKalman;
        void check_coordinate(uint8_t k,uint8_t i);
        void present_candidate(uint8_t k);
        void new_candidate(uint8_t k);
        void evaluate_candidate(uint8_t k);
        void downgrade_dissociated_vector();
        logger_txt logger_StatusVector_Control;
        char log[100],file_logger[100];

    public:
        StatusVectorControl();
        void          set_Coordinate_aux_vect_pointer          (uint16_t**
pointer_Coordinate_aux_vect);
        void set_Coordinate_pointer (uint16_t*** pointer_Coordinate);
        void set_VectorEstado_pointer (uint16_t* pointer_VectorEstado);
      void set_Kalman_medidas_pointer (uint16_t*** pointer_Kalman_medidas);
        void set_VectorEstadoKalman(uint16_t VectorEstadoKalman);
        void set_cont_seq(uint32_t cont_seq_StatusVector);
        void  set_Coordinate  (uint16_t  coor_x,  uint16_t  coor_y,  uint16_t
coor_z);
        uint16_t get_VectorEstadoKalman(){ return VectorEstadoKalman;}
```

```
      uint8_t get_new_userID(){ return new_userID;}
        uint16_t check_StatusVectorControl(uint8_t medida);
        uint16_t downgrade_active_vector();
};
```

Archivos: StatusVectorControl.cpp

StatusVectorControl.hpp

Clases y métodos:

```
class Actions{
    private:
        int16_t walk_slow_down,walk_slow_up,walk_fast_down,walk_fast_up;
        int16_t umbral_z;
        int16_t x_vel,y_vel,xy_vel,z_vel,z_vel_prev;
        int16_t xy_vel_lp,z_vel_lp;
        uint32_t count_aux;
        uint32_t count_seq;
        uint32_t frame_seq;
        uint8_t state_user[16];
        int16_t xy_vel_hist[16][5],z_vel_hist[16][5];
        uint16_t*** Kalman_medidas;
        uint16_t* VectorEstado;
        uint8_t flag_user_alone;
        int cont_user;
        void evaluate_state_user(uint8_t id);

        statics_csv statics_actions_speed;
        logger_txt logger_actions_speed;
        char log[100],file_logger[100];
    public:
        Actions();
        Actions(char name_file_csv[100]);
        void calculate_speed();
        void close_files();
        void no_user_frame();
        void set_cont_seq(uint32_t seq);
        void set_cont_user(int cont_user_extern);
        void clean_Kalman_medidas(uint8_t id);
```

```
          void set_frame( uint32_t frame_local );
          void            set_Kalman_medidas_pointer            (uint16_t***
pointer_Kalman_medidas);
          void set_VectorEstado_pointer (uint16_t* pointer_VectorEstado);
          uint8_t* get_state_user(){ return state_user; }
   };
```

Archivos: Utilities.cpp

Utilities.hpp

Statics.py

Clases y métodos:

```
   class logger_txt{
        private:
           FILE * pointer_file_logger;
           uint8_t level_logger;
       public:
             logger_txt();
           logger_txt(char file_logger[100]);
           void write_log(char log[100],uint8_t level_log);
           void set_level_logger (uint8_t level_value);
   };

   class statics_csv{
        private:
           FILE * file_state;
           FILE * file_vel;
           char file_logs_path[100], info_csv[100];
           uint32_t frame;
       public:
             statics_csv();
           statics_csv(char file_logger[100]);
           void write_vel(uint8_t xy_vel, uint8_t z_vel);
           void write_state(uint8_t status_action);
           void set_frame (uint32_t local_frame);
           void close_file();
   };
```