UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería de Computadores

Trabajo Fin de Grado

SmartPoliTech Body Tracker

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería de Computadores

Trabajo Fin de Grado

SmartPoliTech Body Tracker

Autor: Juan Pedro Torres Muñoz

Tutor: Antonio M. Silva Luengo

ÍNDICE GENERAL DE CONTENIDOS

Contenido

1 INTRODUCCIÓN 8

2 OBJETIVOS 8

3 METODOLOGÍA. ESTADO DEL ARTE 8

3.1 Metodología de trabajo 9

3.2 Programación basada en componentes 9

3.3 Frameworks 10

3.3.1 JdeRobot 10

3.3.2 Robocomp 10

3.3.3 ZeroC ICE 11

3.4 Tecnología kinetica 11

3.4.1 Cámaras disponibles en el mercado 11

3.4.2 Elección de cámaras para la realización del trabajo 13

3.5 SmartPoliTech 14

3.6 Python 15

3.7 Cálculo de trayectorias 15

3.7.1 Filtro de Kalman 16

4 IMPLEMENTACIÓN Y DESARROLLO 17

4.1 Planteamiento inicial 17

4.2 División del componente 19

4.2.1 Diseño de las Interfaces 20

4.2.2 Generación IDSL’s 23

4.2.3 Generación de los componentes 23

4.2.4 Problema componente CameraReader 25

4.3 Método lectura de las cámaras 28

4.4 Diseño del componente hallMonitor 29

4.4.1 Diseño e implementación clase Individuo 29
4.4.2 Algoritmo de actualización de individuo 31
4.4.3 Problema en esta versión 32
4.4.4 Estimación de posiciones 33
4.4.5 Condiciones para eliminar un individuo del sistema 36
4.5 Pruebas y problemas 37
4.6 Solución al problema 38
4.7 Problemas encontrados con esta solución 39
4.8 Solución al problema 39
4.9 Problemas encontrados con esta solución 40
4.10 Solución al problema y estado final del sistema 41
4.11 Script de visualización 42
4.11.1 Generación del código fuente 42
5 RESULTADOS Y DISCUSIÓN 43
5.1 Seguimiento de una persona 43
5.2 Seguimiento de dos personas 44
5.3 Problemas y alternativas 45
6 CONCLUSIONES 46

ÍNDICE DE TABLAS

Tabla 1. Comparativa entre diferentes sistemas RGBD..... 12

ÍNDICE DE FIGURAS

Ilustración 1.Representación genérica de componentes.[2] 9
Ilustración 2. Mapa Pabellón Señalizando las cámaras con puntos rojos. 13
Ilustración 3. Logo SmartPoliTech.[4] 14
Ilustración 4. Fases filtro Kalman 17
Ilustración 5. Primera aproximación del Tracker. 19
Ilustración 6. Aproximación con dos componentes. 20
Ilustración 7. Representación flujo CameraReader 28
Ilustración 8. Algoritmo inicial hallMonitor 29
Ilustración 9. Diagrama UML de clase Individuo. 30
Ilustración 10. Diagrama flujo actualizar individuo. 31
Ilustración 11. Diagrama de flujo tras las primeras pruebas. 33
Ilustración 12. Prueba funcionamiento Pykalman perdiendo 1 / 5 de los datos 34
Ilustración 13. Prueba funcionamiento Pykalman perdiendo 4 / 5 de los datos 35
Ilustración 14. Diagrama de flujo updateKalman. 36
Ilustración 15. Diagrama de flujo del metodo de borrado de individuos. 37
Ilustración 16. Diagrama de flujo modificado para borrar dos personas. 38
Ilustración 17. Diagrama de flujo con borrado de cada cámara. 40
Ilustración 18. Diagrama de flujo final del componente hallMonitor. 41
Ilustración 19.Diagrama de flujo readMonitor. 43

RESUMEN

Este proyecto se ha enfocado en la implementación de un sistema de reconocimiento, etiquetado y seguimiento de personas de forma totalmente anónima. Para ello se ha realizado una implementación del proyecto basado en componentes, implementados en Python, permitiéndonos reutilizar dichos componentes en caso de necesitarlos. Para ello se ha hecho uso de las cámaras del proyecto SmartPoliTech presentes en el pabellón.

Este sistema nos permite conocer el posicionamiento y realizar el seguimiento de una persona dentro de la zona acotada sin perdidas. Todo gracias a la predicción de posiciones realizada en caso de entrada del individuo en una zona sin visión de las cámaras. También se pone a disposición una interfaz ICE para la visualización de los datos ya procesados.

ABSTRACT

This project has focused on the implementation of a system of recognition, labeling and monitoring of people in a totally anonymous way. For this, we have implemented a component-based implementation in Python, allowing us to reuse those components in case they are needed. The cameras of the SmartPoliTech project present in the pavilion have been used for this purpose.

This system allows us to know the position and follow a person within the bounded zone without losses. All thanks to the prediction of positions made in case of entry of the individual into an area without vision from the cameras. An ICE interface is also available for displaying the data already processed.

1 INTRODUCCIÓN

Debido al auge de las ciudades inteligentes, se nos plantea la necesidad de conocer y llevar un seguimiento de las personas presentes en cierto espacio, ya sea una casa, una tienda, un hotel o cualquier recinto que pueda sufrir grandes cambios en la cantidad de personas presentes.

Todo este proceso debe ser realizado de forma anónima, ya que en ningún momento se pretende invadir la privacidad de las personas, pues los datos que queremos, serán utilizados de manera estadística para automatizar diferentes sistemas (temperatura, luces, etc…). Pudiendo obtenerse la información del número de personas presentes mediante diferentes técnicas.

2 OBJETIVOS

Este trabajo tratará de construir un sistema que nos dote de la capacidad para reconocer, etiquetar de forma única y realizar un seguimiento en tiempo real, de las personas presentes dentro de una zona acotada.

Para la elaboración de este sistema se utilizarán las cámaras kineticas presentes en el pasillo del pabellón de informática, pertenecientes al proyecto SmartPoliTech. Con la información procedente de dichas cámaras y con el objetivo de mantener el conocimiento de la existencia de cada individuo de forma única, el sistema propuesto deberá tener en cuenta el movimiento de dichos individuos, para eso se ha de implementar un sistema de predicción de trayectorias, además del propio sistema para reconocer que individuo es cual en cada cámara.

3 METODOLOGÍA. ESTADO DEL ARTE

En esta sección se hablará tanto de la metodología utilizada para desarrollar el proyecto, como las metodologías utilizadas para resolver ciertos problemas, como a su vez también se describirá el estado del arte, incluyendo el paradigma de programación utilizado para desarrollar la solución propuesta, del framework de programación con el que se ha realizado la integración del sistema, de las cámaras kineticas presentes en el mercado, de la tecnología kinetica en general, de las diferentes técnicas de seguimiento que existen hasta el momento.

3.1 Metodología de trabajo

Para el desarrollo del trabajo no se definió una metodología inicialmente, aunque se ha trabajado lo que en el papel vendría a ser una modificación de una metodología "AGILE". Dichas metodologías se centran en el desarrollo de prototipos funcionales sobre los cuales se trabaja de manera incremental, como se ha trabajado en este proyecto, otro detalle importante han sido las reuniones, no diarias como está pensado originalmente en Scrum[15] pero se han sucedido según iban siendo necesarias, para aclarar las dudas surgidas en lo referente a funcionamiento, implementación o alguna técnica para mejorar el funcionamiento.

De la explicación anterior, podemos deducir que hemos adaptado Scrum a nuestro problema y a nuestra situación.

3.2 Programación basada en componentes

En primer lugar hemos de describir que se entiende como componente en este paradigma de programación. De ahora en adelante, un componente será un fragmento de código que implementará la función deseada, siendo una pieza de código estanca, dedicada únicamente a realizar su función específica, robusta y fácilmente reutilizable por otro programador sin tener que preocuparse por seguridad, fiabilidad y tolerancia a fallos. Dichos componentes exponen una interfaz de comunicación a través de la cual se comunican con otros componentes.[2][23]

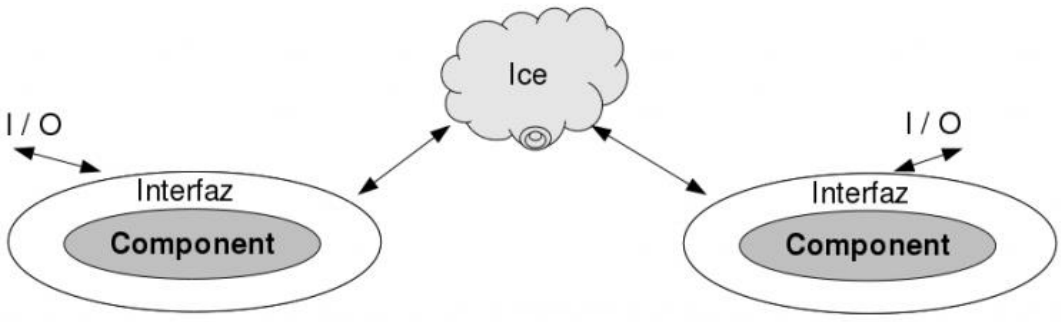

Ilustración 1.Representación genérica de componentes.[2]

Como se comenta en el artículo [23], la programación basada en componentes está en auge, y es de gran aplicación en la industria. Pero nos encontramos con el

problema de que para poder aprender esta técnica de programación ha de comprenderse bien la programación orientada a objetos. Debe hacerse hincapié en que la parte más importante de los componentes es el diseño de la interfaz de comunicación y de la posibilidad de reutilización del componente sin tener que modificar nada.[24]

3.3 Frameworks

Un framework es un entorno de desarrollo que nos proporciona una forma de desarrollar y de desplegar nuestro software. Los frameworks incluyen librerías de soporte, ejemplos, herramientas para facilitar las tareas más habituales, tales como la generación de código, cálculos habituales, etc…, ofreciéndonos también APIs para trabajar con los componentes propios del framework o los creados por nosotros mismos.

3.3.1 JdeRobot

Este framework de desarrollo centrado en la robótica y en la visión artificial, debido a que nuestro proyecto va a servirse de cámaras kineticas, se planteó la opción de poder realizarse con este framework, el cual nos permitía usando una interfaz ICE. [1]

Siendo posible generar el código y trabajar usando este framework, se descartó la idea de utilizar este framework, debido a que no tenía gran conocimiento de la estructura del framework, dejando de lado esta alternativa, siendo lo único positivo que no hacía falta que se adaptara la interfaz que nos proporcionaba la empresa que instaló las cámaras.

3.3.2 Robocomp

Framework open-source, centrado como el anterior en la robótica. La comunicación entre los componentes desarrollados utilizando este framework tiene lugar utilizando interfaces del framework ICE, igual que el framework anterior.

Las ventajas para la utilización de este framework son las siguientes: conocimiento de la estructura del framework debido a que ya se ha trabajado con él

durante el curso, facilidad para la generación de componentes gracias a su generador de código, y la total compatibilidad para realizar el proyecto en Python. El único inconveniente que se nos presenta es que este framework no nos permite generar código usando la interfaz directamente provista por el vendedor.[2]

3.3.3 ZeroC ICE

Framework de comunicación, usado por los frameworks citados anteriormente, que nos permite establecer comunicaciones RPC entre nuestros componentes, usando los ficheros ICE en los que se establece el acuerdo de la interfaz a usar. Tiene una implementación nativa para muchos lenguajes (C++, C#, Java, JavaScript, Python, Ruby, etc…). También nos proporciona un servidor de mensajes para el caso de que no queramos usar RPC, en este caso no se utilizará.[3]

Como utilizaremos Robocomp como framework de desarrollo todo el código relativo a la carga, implementación y creación de los proxies de comunicación será generada, por lo que prácticamente no tendremos que preocuparnos por ella.

3.4 Tecnología kinetica

También conocidos como sensores RGB-D, las cuales son capaces de captar las bandas del color, además de guardar la información de la profundidad para cada pixel de la imagen.

Esta tecnología ha sido desarrollada por PrimeSense[5] y patentada en Estados Unidos. Esta tecnología esta solo disponible comercialmente para Microsoft Kinect y Asus Xtion. En este tipo de cámaras una vez calibrado el sistema RGB con el que obtenemos la imagen de color, simultáneamente un sensor infrarrojo calibrado para que cada "pixel" de la imagen de profundidad corresponda con el pixel especifico de la imagen a color. Con esto podemos generar una nube de puntos, con la que podemos asignar cada punto de la imagen en un espacio tridimensional.

3.4.1 Cámaras disponibles en el mercado

Actualmente, se supone que existe una patente[7] vigente, pero aparte de las citadas anteriormente Microsoft Kinect y Asus Xtion, tanto la versión Live y la

versión PRO. Aunque, he podido encontrar otros modelos disponibles en la red, que son los siguientes modelos:

- Intel RealSense R200[10]
- Stereolabs ZED[6]
- Carnegie Robotics MultiSense S7[11]
- e-Con Systems Tara Stereo Camera[12]
- Narian SPI[13]

A continuación podremos ver una tabla comparativa entre las cámaras anteriormente citadas:

Tabla 1. Comparativa entre diferentes sistemas RGBD

	Resolución Máxima	Campo de Visión	Profundidad	Consumo Energético
Microsoft Kinect 2[8]	1920x1080	70º Horizontal 60º Vertical	4.5 metros	NA
Asus Xtion[9]	640x480	58ºHorizontal 45º Vertical	3.5 metros	~2.5W
Intel RealSense R200	1920x1080 color 480x360 profundidad	70º Horizontal 59º Vertical	3.5 metros	~1.6W
Setereolabs ZED	4416x1242@15fps 3840x1080@30fps	110º Horizontal	20 metros	~1.9W
Carnegie Robotics Multisense S7	1024x544@30fps 2048x1088@7fps	80º Horizontal 49º Vertical (2MP) 80º Vertical	NA	~7W

		(4MP)		
e.Con Systems tara Stereo Camera	752x480@60fps	60º Horizontal	3 metros	NA
Narian SPI	640x480@30fps	43.6° Horizontal 33.4º Vertical	NA	~4W

(NA: Información no disponible)

3.4.2 Elección de cámaras para la realización del trabajo

Una vez vistos los datos de Tabla 1, se debe seleccionar un modelo de cámara para llevar a cabo la implementación de nuestro trabajo, por desgracia, y pese a haber muy buenas alternativas presentes, el proyecto ha tenido que realizarse con las cámaras que ya habían sido instaladas en el pabellón por parte de SmartPoliTech.

Son unas cámaras muy parecidas a la Asus Xtion, pero con ciertas mejoras, pues pueden llegar a cerca de 8 metros de profundidad, suficiente para cubrir prácticamente el ancho completo del pasillo en el que vamos a trabajar.

El sistema con el que vamos a trabajar consta de cuatro NUCs con 2 cámaras conectadas cada uno, con lo que se cubre un espacio de unos 30 metros, instalados de forma alterna en el pasillo a estudiar de la siguiente manera.

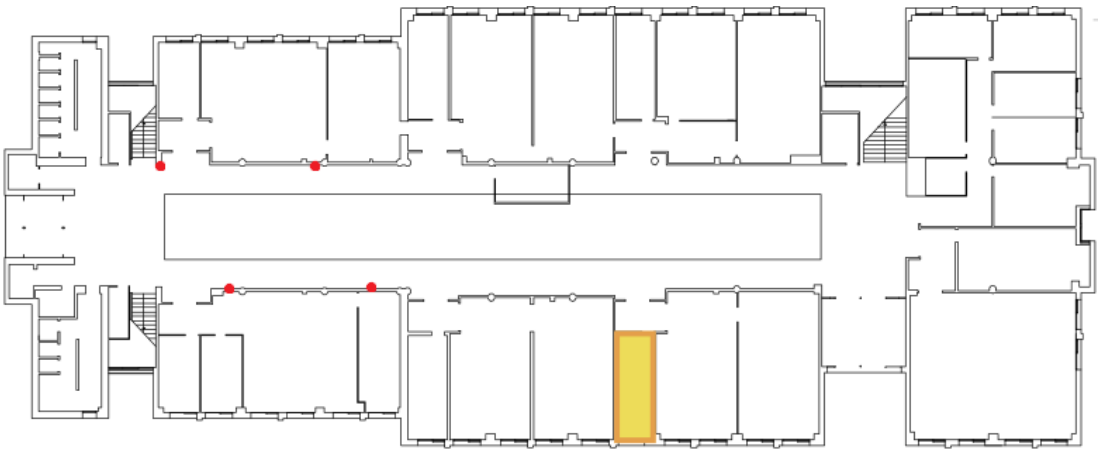

Ilustración 2. Mapa Pabellón Señalizando las cámaras con puntos rojos.

3.5 SmartPoliTech

Proyecto que intenta convertir a la escuela politécnica de Cáceres en un "living-lab", un lugar donde la tecnología contribuye al bienestar de las personas, el ahorro energético y favorecer la investigación. El objetivo es poder diseñar, implementar, integrar y validar sistemas capaces de crear espacios inteligentes.[4]

El espacio ha ido cambiando desde un espacio tradicional hasta el objetivo de un espacio inteligente con el que los habitantes puedan interactuar. Basándose en la filosofía OpenData, para permitir que cualquiera pueda utilizar dichos datos para realizar el proyecto que se le ocurra. Esto genera que haya que desarrollar la tecnología en cinco áreas diferentes: conectividad de sensores, almacenamiento de datos, visualización, modelado y planificación. Todo esto nos lleva a interactuar con diferentes tecnologías emergentes como pueden ser, IoT, NoSQL, tecnologías cloud, minería de datos y algunas más.[4]

El impacto final inmediato del proyecto es el ahorro energético que supone, ya que por ejemplo no hay que mantener las luces encendidas constantemente, pues solo se encenderán en el momento en el que haya gente presente. Además se intenta generar un ecosistema en el que grupos interesados en dichas tecnologías pueden colaborar con los diferentes grupos de investigación que están creando el espacio inteligente. Siendo de esta manera un lugar de referencia para todas esas ciudades que están interesadas en el desarrollo de las Smart Cities, las cuales están tomando fuerza en este momento[4].

Ilustración 3. Logo SmartPoliTech.[4]

3.6 Python

Python será el lenguaje utilizado para la implementación del proyecto por lo que parece necesario comentar algo sobre él, ya que no se ve en profundidad en la carrera.

Python es un lenguaje de programación que fue creado en 1991, interpretado, por lo que no necesita ser compilado, con lo que nos va a dar mucho juego al poder realizar las pruebas desde cualquier máquina y la reusabilidad de nuestros componentes una vez terminados. También nos encontramos con un lenguaje multiparadigma, pudiendo programarse usando programación orientada a objetos, programación imperativa o programación funcional. Cuenta con tipado dinámico, por lo que las variables no tienen tipo definido hasta el momento en el que se usan. Otro detalle curioso es que en este lenguaje la organización del texto se basa en indentaciones del código para agruparlo. Cuenta además con tipos de datos de alto nivel como pueden ser listas, diccionarios y conjuntos.

Posee una licencia "open-source" y esta administrado por la Python Software Foundation. Hace hincapié en el concepto de la legibilidad del código pues un buen código en este lenguaje debería ser como leer en inglés.[14]

Debido a su legibilidad, portabilidad, sus tipos de alto nivel, la integración e interacción de Robocomp con dicho lenguaje y la existencia de librerías para poder trabajar utilizando las interfaces de ZeroC ICE con este lenguaje lo convierten en la mejor opción para desarrollar el proyecto.

3.7 Cálculo de trayectorias

Aquí explicaremos las diferentes formas con las que se podría inferir la posición de una persona en un plano 2D, siendo este uno de los problemas que vamos a tener que resolver en el proyecto.

Para ello vamos primero a separar en dos posibles casos, movimiento rectilíneo uniforme (m.r.u), el movimiento rectilíneo uniformemente acelerado (m.r.u.a) y por último el movimiento rectilíneo acelerado (m.r.a). Como nos encontramos en un situación en la que tendremos personas caminando, podemos asegurar que no nos encontraremos ante un movimiento rectilíneo uniformemente acelerado, ya que las personas presentes en nuestro entorno pueden variar la aceleración a la que se están

moviendo, ya sea porque vayan a detenerse a hablar con alguien o aligeren el paso para no llegar tarde a clase. En ambos ejemplos se modificaría la aceleración por lo que podemos descartar el movimiento el movimiento rectilíneo uniformemente acelerado y el movimiento rectilíneo uniforme, ya que en este último no habría cambios de velocidad, lo cual al modelar el movimiento de una persona ya produce desconfianza al asumir que varias personas se moverán a la misma velocidad sin alterarla.

Con lo anterior nos deja con que la ecuación que refleja el movimiento de las personas para este caso es un movimiento rectilíneo acelerado, ahora nuestro problema se limita a obtener la aceleración en cada instante o a calcular la velocidad en cada instante. Como no tenemos forma de medir la aceleración en nuestro sistema, vamos a optar por calcular una velocidad inmediata estimada del incremento de velocidad respecto a la medida de velocidad anterior, que será con la que calculemos el siguiente punto en el espacio.

$$V_{t+1} = \Delta V$$

$$Pos_{t+1} = Pos_t + \Delta T * V_{t+1}$$

Estas serán las ecuaciones, con las que calcularemos las posiciones en caso de que no tengamos medida de ninguna cámara de la persona en concreto. Para realizar esta tarea se va a utilizar una implementación del filtro de Kalman.

3.7.1 Filtro de Kalman

Este filtro se compone de una parte de dos partes, en la primera parte se hace una predicción del estado futuro del sistema, tras obtener la medida real, se calcula la varianza respecto a la medida estimada y se corrige el sistema, para que en siguientes estimaciones sean lo más correctas posibles.[22]

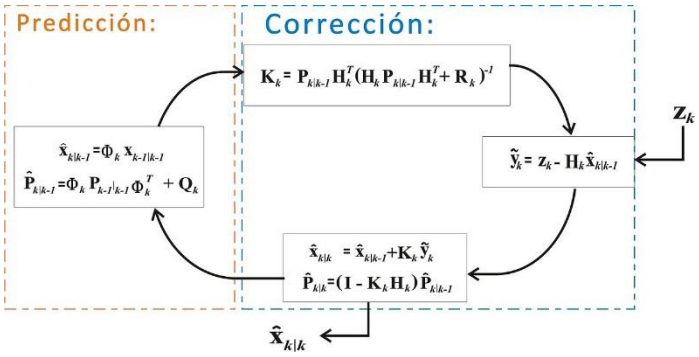

Ilustración 4. Fases filtro Kalman

4 IMPLEMENTACIÓN Y DESARROLLO

En esta sección se va a describir los pasos seguidos, las decisiones de diseño llevadas a cabo durante el desarrollo, los cambios en el diseño y las razones por las que estos han tenido lugar, también se explicarán los algoritmos más importantes que tienen lugar en el proyecto, junto con las diferentes técnicas con las que se han intentado solucionar problemas y con las que se han solucionado realmente los problemas.

4.1 Planteamiento inicial

El primer acercamiento a la solución fue conseguir que un script en Python pudiera leer de las cámaras y mostrarnos la información que nos enviaba dicha interfaz. Primero vamos a explicar la interfaz que usamos para comunicarnos con los componentes, puesto que no hemos hablado de ella todavía.

```
#ifndef PEOPLETRACKER_ICE
#define PEOPLETRACKER_ICE
module peopleTracker3dMod {
        struct PlayerPos{
            float x;
            float y;
            float z;
        };
        sequence<PlayerPos> Positions;
        struct PersonInfo{
                Positions pos;
                PlayerPos size; //volume
                bool predicted;
                long id;
        };
        sequence<PersonInfo> People;
        struct peopleData{
                People data;
        };
        interface peopletracker3d{
                idempotent peopleData getData();
        };
};
#endif
```

Como podemos ver, la interfaz nos ofrece una estructura que contiene una lista con los datos de cada persona en dicha cámara. Estos datos estarán formados por un id único para esa persona mientras este en dicha cámara, una lista con las ultimas “n” estimaciones realizadas, un campo que representa el tamaño en cada dimensión para que podamos obtener el volumen de la persona y un valor para indicar si la última posición de esa persona es medida o estimada con los datos anteriores.

Una vez conseguimos realizar la lectura de una cámara, se continuó a diseñar la estructura de nuestro proyecto, siendo la siguiente:

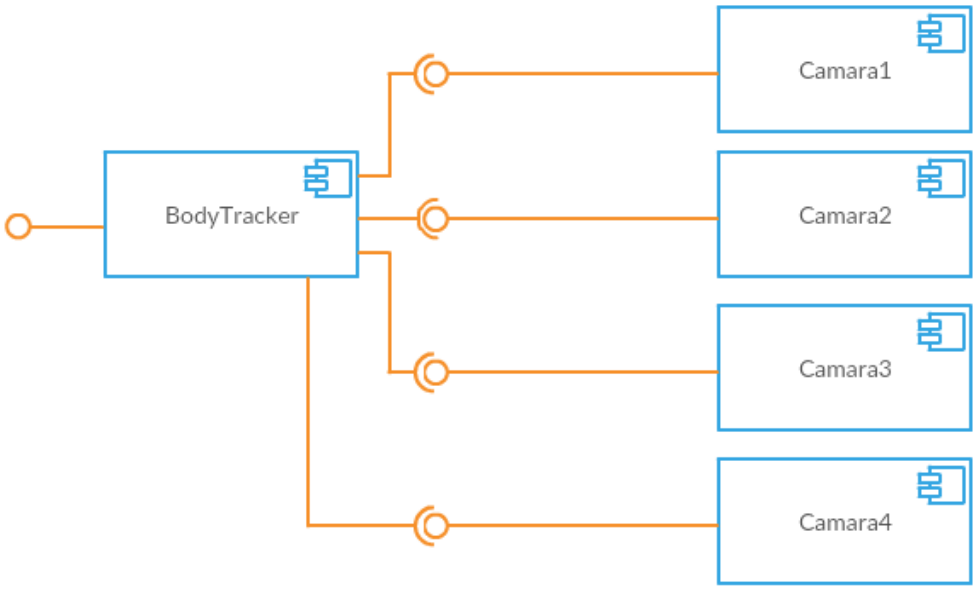

Ilustración 5. Primera aproximación del Tracker.

En el primer planteamiento se intentó tener solo un componente para que realizara la lectura de las cuatro cámaras, la selección de los datos que nos interesaba de todos los mandados, el posterior cálculo para asignar el id a cada persona y el seguimiento consecuente, aparte de proveer una interfaz a través de la cual se podría consultar el estado del sistema, las personas presentes con sus respectivo id único para todo el tramo observable del pasillo.

Tras intentar realizar la primera implementación se pudo comprobar que había demasiado solapamiento de las tareas a realizar en dicho componente, creando un componente que no podría ser reutilizado nada más que para leer de las y realizar el tracking de las personas en ese tramo.

4.2 División del componente

Se nos presentaba el problema contado anteriormente, de que se acumulaban tareas a realizar en el componente, lo que se convertía en un sistema ralentizado, visto esto se optó por cambiar el diseño del sistema al siguiente:

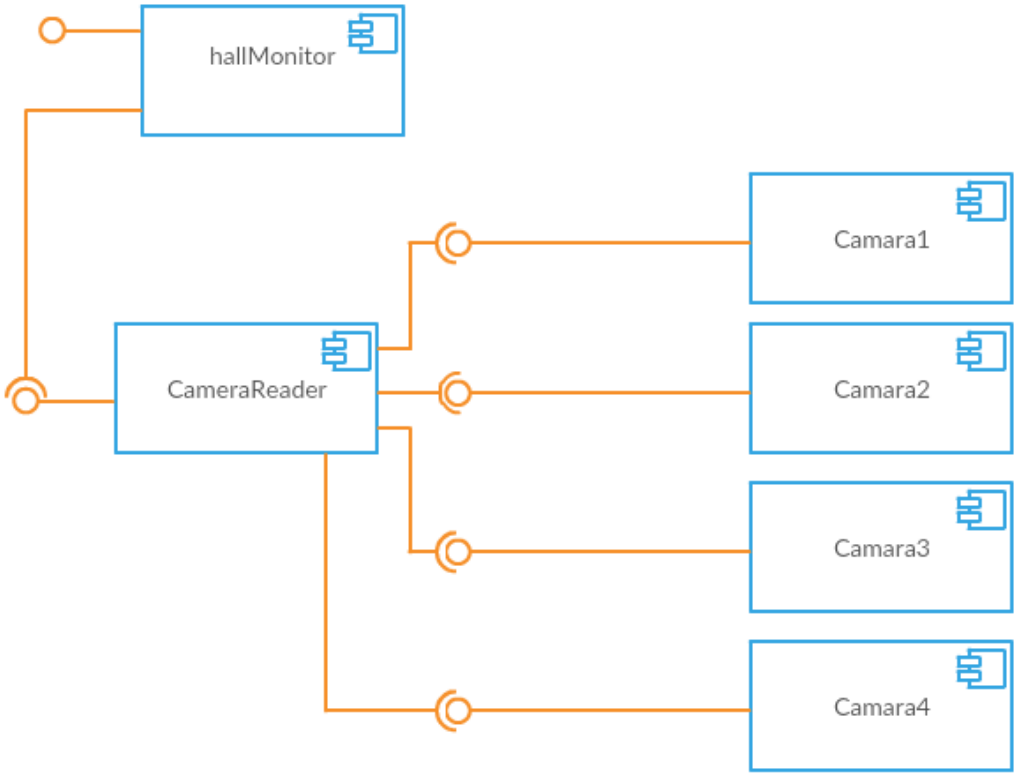

Ilustración 6. Aproximación con dos componentes.

Una vez visto que no vamos a tener todo el proyecto en un único componente, se descarta también la opción de tenerlo en un único script de Python como estaba hasta ahora, y se empezará a usar Robocomp como framework de desarrollo del proyecto. Como ahora vamos a tener varios puntos de comunicaciones entre componentes, hemos de desarrollar una interfaz ICE para la conexión entre el componente CameraReader (componente encargado de la lectura de todas las cámaras) y otra para la interfaz que ofrece el propio hallMonitor (componente encargado de realizar el seguimiento de las personas) para así poder ver el estado del sistema.

4.2.1 Diseño de las Interfaces

Para empezar vamos a diseñar la interfaz de comunicación entre el componente CameraReader y el componente hallMonitor. Para ello debemos pensar en que datos de los que leemos de las cámaras necesitamos realmente para realizar el tracking.

En primer lugar, y como este componente solo tiene la misión de leer de todas las cámaras disponibles del sistema, ya sean las cuatro disponibles en nuestro sistema, o un número diferente de cámaras en otro sistema, configurando esto al generar el componente, pero hablaremos de esto luego.

Los datos a recibir serán los siguientes: una lista de personas, cada persona será una estructura que contendrá la información para identificar a la persona, los cuales son posición, id de la cámara en la que se encuentra la persona, el id asignado por la cámara a la persona, un identificador para saber si la posición proporcionada es estimada o real, por ultimo también utilizaremos las dimensiones de la persona, para posteriormente utilizarlas para calcular el volumen, el cual se utilizará en esta aproximación a la solución. Quedando la interfaz como sigue:

```
#ifndef HALLSTATUSCOMP_ICE
#define HALLSTATUSCOMP_ICE
module hallStatusComp
{
	struct PlayerPos{
		float x;
		float y;
		float z;
	};
	struct PersonInfo{
		PlayerPos pos;
		bool predicted;
		PlayerPos vol;
		long id;
		int idCam;
	};
	sequence <PersonInfo> People;
	struct hallData{
		People data;
	};
	interface peopleHall
	{
		idempotent hallData getData ();
	};
};
#endif
```

Ahora tendremos que diseñar la interfaz que ofrecerá el componente hallMonitor para que se puedan leer los datos del sistema y mostrarlos en un visualizador.

Para esta interfaz necesitaremos ofrecer una lista con la información de cada persona, la única información que se dará de cada persona será un identificador único para cualquier persona en la zona y una posición en el pasillo para esa persona, quedándonos la siguiente interfaz:

```
#ifndef READHALLCOMP_ICE
#define READHALLCOMP_ICE
module readHallComp
{
	struct Pos{
		float x;
		float y;
		float z;
	};
	struct PersonInfo{
		Pos pos;
		int id;
	};
	sequence <PersonInfo> People;
	struct hallPersons
	{
		People data;
	};
	interface readHall
	{
		idempotent hallPersons getHall ();
	};
};

#endif
```

Una vez tenemos definidas ambas interfaces, nos dispondremos a generar los ficheros IDSL respectivos de cada interfaz para poder así usar el generador de código de Robocomp.

4.2.2 Generación IDSL's

Como nos encontramos utilizando Robocomp como framework de desarrollo vamos a aprovechar su herramienta de generación de código, la cual también sirve para generar ficheros IDSL.

Para ello solo tenemos que irnos a la carpeta contenedora de nuestros ficheros .ice y utilizando la consola de Linux (teniendo instalado correctamente Robocomp) utilizar el siguiente comando.

```
robocompdsl nombreInterfaz.ice nombreFichero.idsl
```

Una vez tengamos generados los ficheros IDSL, moveremos a la carpeta "interfaces" de nuestra instalación de Robocomp, para así poder utilizar dichos IDSL a la hora de generar los nuevos componentes.

4.2.3 Generación de los componentes

Se va a hacer una breve introducción a la generación de componentes usando el framework Robocomp, también podemos encontrar una guía de cómo utilizar el generador de componentes en el propio repositorio de Robocomp[16].

Crearemos una carpeta vacía donde estará nuestro componente, usaremos el siguiente comando, el cual nos generará un esqueleto de fichero CDSL que tendremos que editar para generar el componente.

```
robocompdsl camReader.cdsl
```

Con esto tendremos que modificar el fichero esqueleto generado para que cumpla las necesidades que debe cumplir nuestro componente. Quedando el siguiente fichero:

```
import "/robocomp/interfaces/IDSLs/peopleTracker3d.idsl";
import "/robocomp/interfaces/IDSLs/hallStatus.idsl";

Component cameraReader
{
        Communications
        {
                implements peopleHall;
                requires peopletracker3d, peopletracker3d, peopletracker3d, peopletracker3d;
        };
        language Python;
};
```

Se explicará un poco la sintaxis del CDSL, en este fichero podemos ver como incluimos los IDSL con el código de las interfaces ICE en las primeras dos líneas, posteriormente definimos las comunicaciones que tendrá nuestro componente las cuales son cuatro interfaces de petición de datos, una a cada cámara presente en el pasillo de observación e implementará una interfaz, llamada peopleHall que es la interfaz que se encargará de comunicar a nuestro componente lector de las cámaras con el componente encargado de realizar el seguimiento, control de la gente, la cual ya fue explicada con anterioridad. Por ultimo seleccionamos el lenguaje en el que estará nuestro componente, como ya se explicó será en Python.

Una vez tenemos el fichero configurado, nos disponemos a generar todos los ficheros con el siguiente comando.

```
robocompdsl camReader.cdsl .
```

El cual generará toda la estructura del componente, directorios y código fuente del mismo. Una vez hecho esto ya podemos modificar nuestro componente en el fichero fuente “specificworker.py”.

Con este componente hecho, procederemos a realizar los mismos pasos con el componente hallMonitor, saltándonos ahora la gran mayoría de los pasos, ya que ya fueron explicados una vez.

Para el componente hallMonitor tendremos el siguiente fichero CDSL:

```
import "/robocomp/interfaces/IDSLs/hallStatus.idsl";
import "/robocomp/interfaces/IDSLs/readHall.idsl";

Component hallMonitor
{
        Communications
        {
                requires peopleHall;
                implements readHall;
        };
        language Python;
        gui Qt(QWidget);
};
```

Como podemos ver, el componente implementará la interfaz readHall, la cual esta descrita con anterioridad, y es la encargada de proveer los datos finales del proyecto para su visualización. Tenemos también que hace uso de la interfaz peopleHall que es la que implementará el componente CameraReader, leyendo así los datos ya filtrados de los leídos de las cámaras. El lenguaje utilizado para la implementación de este componente será también Python.

4.2.4 Problema componente CameraReader

En este momento, nos encontramos con que el componente CameraReader no funciona, ya que da una excepción al arrancar. Esto es debido a que en el código generado por robocompdsl a la hora de generar los proxies de conexión de ZeroC ICE se utilizan las siguientes llamadas:

```
try:
        proxyString = ic.getProperties().getProperty('peopletracker3d1Proxy')
        try:
                basePrx = ic.stringToProxy(proxyString)
                peopletracker3d1_proxy = peopletracker3dPrx.checkedCast(basePrx)
                mprx["peopletracker3dProxy1"] = peopletracker3d1_proxy
        except Ice.Exception:
                print 'Cannot connect to the remote object (peopletracker3d)', proxyString
                #traceback.print_exc()
                status = 1
except Ice.Exception, e:
        print e
        print 'Cannot get peopletracker3dProxy property.'
        status = 1
```

Esa llamada, para generar el proxy de conexión hace una comprobación entre la interfaz que tiene el cliente y la interfaz que tiene el servidor, viniendo aquí nuestro problema, pues la interfaz que tenemos en el cliente es una modificación para que robocompdsl pueda generarnos el código fuente, ya que no es capaz de generar código cuando en los ficheros se utilizan clases en vez de estructuras. En el siguiente cuadro de texto se puede ver como se usa la palabra reservada “class” donde antes usábamos la palabra “struct”, este cambio no afecta al funcionamiento de la interfaz, pero si nos permite generar componentes con el generador de Robocomp.

```
module pt3d {
        struct PlayerPos{
                …
        };
        sequence<PlayerPos> Positions;
        struct PersonInfo{
                …
        };
        sequence<PersonInfo> People;
        class peopleData{
                People data;
        };
        interface peopletracker3d{
                idempotent peopleData getData();
        };
};
#endif
```

Para solucionar este leve problema, el cual parece grave a priori, solamente tenemos que cambiar la llamada a ZeroC, a una en la que no se realice la comprobación de las interfaces entre el cliente y el servidor. Quedando el siguiente código:

```
        try:
                proxyString = ic.getProperties().getProperty('peopletracker3d1Proxy')
                try:
                        basePrx = ic.stringToProxy(proxyString)
                        peopletracker3d1_proxy = peopletracker3dPrx.uncheckedCast(basePrx)
                        mprx["peopletracker3dProxy1"] = peopletracker3d1_proxy
                except Ice.Exception:
                        print 'Cannot connect to the remote object (peopletracker3d)', proxyString
                        #traceback.print_exc()
                        status = 1
        except Ice.Exception, e:
                print e
                print 'Cannot get peopletracker3dProxy property.'
                status = 1
```

Con esto tendríamos funcionando correctamente el componente CameraReader por lo que podemos empezar a trabajar en el componente.

4.3 Método lectura de las cámaras

Una vez tenemos el componente funcionando, nos tocará diseñar el algoritmo que utilizaremos para leer de todas las cámaras y filtrar los datos para poder generar una lista con todos los datos que necesitas. El siguiente diagrama de flujo muestra cómo se generará la lista que devolveremos a través de la interfaz ICE.

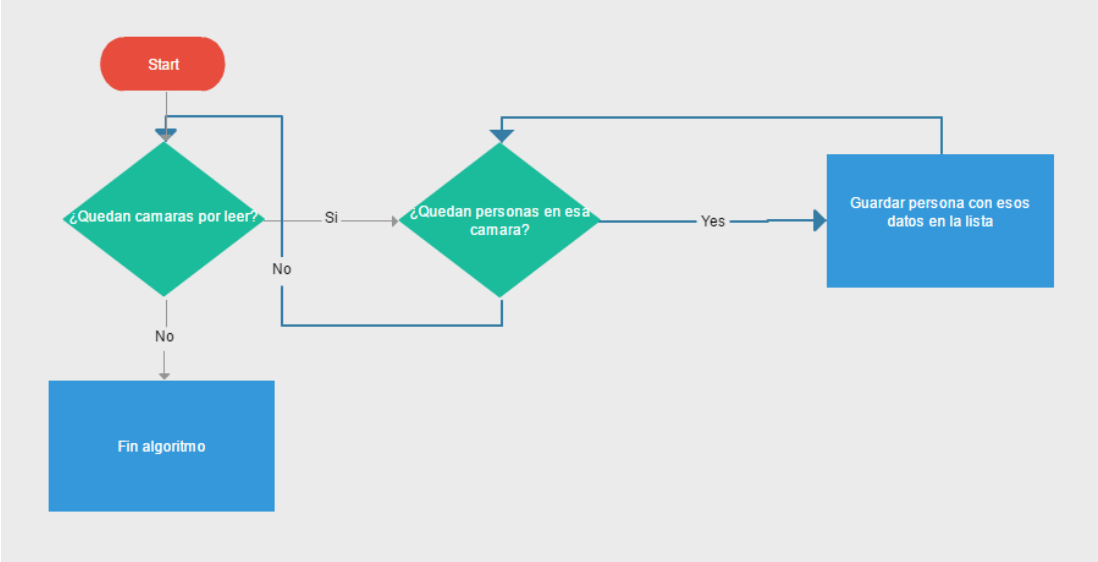

Ilustración 7. Representación flujo CameraReader

Como se puede ver en el diagrama vamos a leer de cada cámara, y para cada persona en cada cámara se guardaran sus datos en la lista, añadiéndole también el identificador de la cámara en la que se encontraba la persona, la última posición de la persona, el identificador dentro de la cámara de esa persona, las tres dimensiones del cuerpo detectado para calcular y una bandera para ver si la posición es estimada o real.

Una vez tengamos el componente realizando esta tarea, este componente estará concluido y podemos avanzar a trabajar con el componente que realizará el seguimiento de todo el pasillo.

4.4 Diseño del componente hallMonitor

El diseño inicial de dicho componente constará de un bucle con el que se recorrerá una lista de personas, las cuales ya están en el sistema, para cada persona en el sistema debe actualizarse, con uno de los datos nuevos de personas recibidos desde el componente CameraReader, una vez hecho actualizar los valores de posición para el generador de posiciones estimadas, y por ultimo eliminar la persona en caso de que se cumplan las características de salida del sistema. Con las personas leídas que no se han utilizado, se crean nuevos individuos en el sistema.

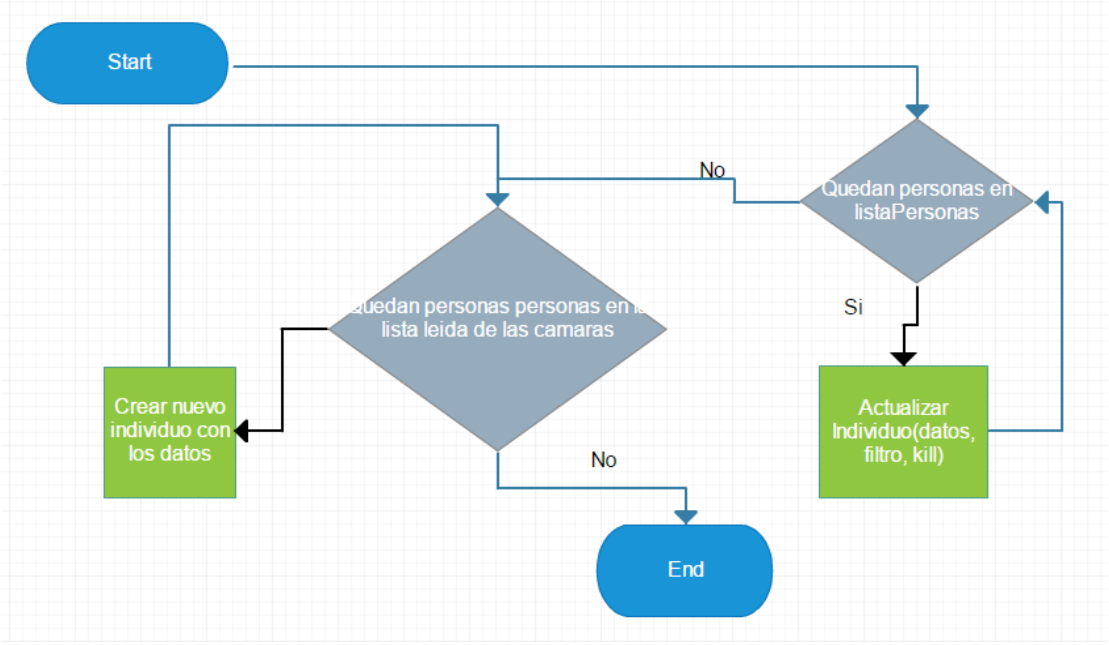

Ilustración 8. Algoritmo inicial hallMonitor

Hasta ahora hemos hablado de actualizar individuos pero todavía no hemos diseñado, ni hablado de la clase que se encarga de gestionar a los individuos dentro del sistema.

4.4.1 Diseño e implementación clase Individuo

Ahora tendremos que diseñar la clase que se encargará de almacenar toda la información necesaria para reconocer, actualizar, estimar posiciones y llevar el cálculo de cuando una persona debe borrarse del sistema. Esta clase estará implementada en Python y será utilizada únicamente en el componente hallMonitor.

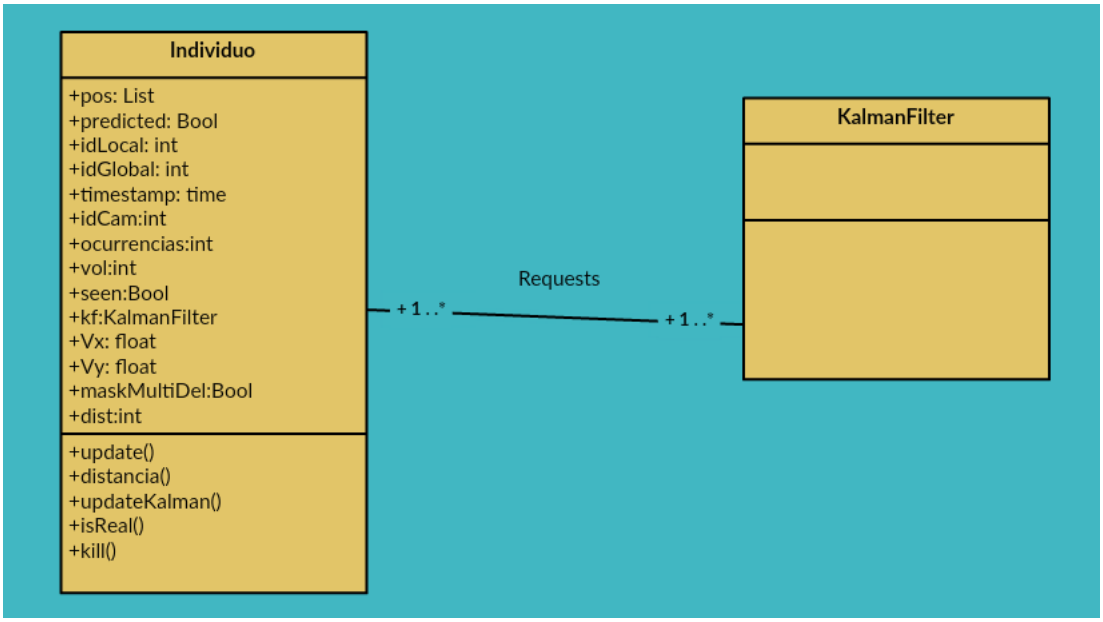

Ilustración 9. Diagrama UML de clase Individuo.

Este es el diagrama UML definitivo de la clase, se explicará a continuación la razón de cada atributo, y su uso, aunque gran cantidad de los atributos de clase son añadidos con posterioridad al planteamiento inicial, para solucionar problemas de funcionamiento posteriores o para añadir funcionalidades que mejoren el funcionamiento del sistema.

Nos encontramos en primer lugar una lista de coordenadas (x,y,z), las cuales son la posición de la persona en el mapa, un booleano llamado predicted que indicará si la posición es estimada, o real, un idLocal que representa el identificador de dicha persona dentro de la cámara en la que se encuentra, un idGlobal que identifica a la persona dentro de nuestro sistema, siendo único, una marca de tiempo de la última vez que se actualizo el individuo con el objetivo de que no permanezcan individuos en el sistema sin ser borrados, un idCam que indica la cámara en la que se encuentra el individuo, ocurrencias que representa el número de veces que se ha actualizado el individuo, vol que es el volumen del individuo, utilizado únicamente en las primeras aproximaciones del sistema, seen que representa si el individuo ha sido actualizado desde una medida de las cámaras, kf que es el filtro de Kalman que se usará para la predicción de posiciones en el caso de que no se encuentre en una cámara la persona, Vx e Vy que representan la velocidad de desplazamiento en el eje x e y respectivamente, maskMultiDel que es un booleano para permitir eliminar a una persona de cada cámara, utilizada en una solución intermedia antes de llegar a la final, y por ultimo dist que guardará el umbral de distancia mínimo para que el

individuo se pueda actualizar. Poco a poco se irán explicando con más detalle a la hora que se vayan introduciendo en el algoritmo de actualización, para solucionar los problemas posteriores.

4.4.2 Algoritmo de actualización de individuo

Para el planteamiento inicial del algoritmo se planteó hacerlo mediante una máquina de estados para gestionar el estado de la persona, y los cambios entre cámaras, pero las condiciones de cambios de estado se complicaban o hasta llegaban a ser imposibles y hubo que buscar otra aproximación.

Para la nueva aproximación se planteó que un individuo se buscará a si mismo dentro de la lista de personas que se acaban de leer del componente CameraReader, una vez ahí, usando el siguiente diagrama de flujo:

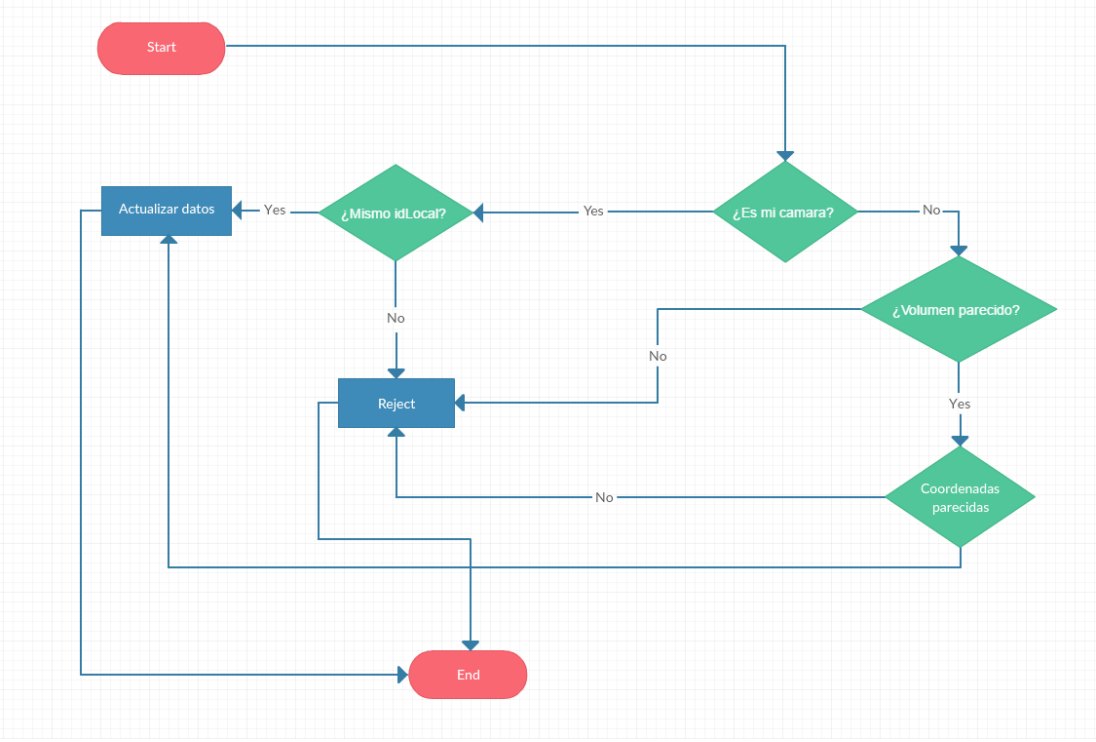

Ilustración 10. Diagrama flujo actualizar individuo.

Este flujo se repetirá para cada persona que tengamos en el sistema, proporcionándole la nueva lista de personas leída desde el componente CameraReader, en la decisión "¿Es mi cámara?" comprobaremos si el idCam del

individuo coincide con el idCam de la persona de la lista nueva que se está procesando, en caso de que si sea la misma cámara se comprueba el idLocal del individuo con el id que trae la persona, en caso de que también sea el mismo, y debido a que la cámara asigna identificadores únicos a cada persona que entra en su rango de visión, podemos asegurar de que es la misma persona, en caso contrario si no es el mismo identificador, podemos asegurar de que no es la misma persona por lo que debe rechazarse. Ahora exploraremos la otra opción, que no sea el mismo identificador de cámara, en cuyo caso en esta implementación comprobamos si el volumen del individuo es aproximadamente el mismo con un margen del 15%, y a partir de ahí en el caso de que no esté en ese rango la persona es descartada o si cumple los valores se comprueba si las coordenadas de esa persona, están dentro del umbral de distancia a la que se ha podido mover nuestro individuo, siendo este valor configurable. Cabe destacar que si se actualiza un individuo con una persona de la lista, dicha persona es eliminada, y en esta versión del algoritmo se termina la ejecución al actualizar al individuo la primera vez.

4.4.3 Problema en esta versión

Tras realizar esta implementación y llevar a cabo las primeras pruebas de funcionamiento, se pudo advertir que la medida del volumen no era eficaz, pues aunque se puede pensar que el volumen de una persona no cambia, esta medida nos la está dando una cámara, por lo que esta medida variará dependiendo del ángulo con el que la cámara nos visualice. Quedando con el siguiente diagrama de flujo el algoritmo.

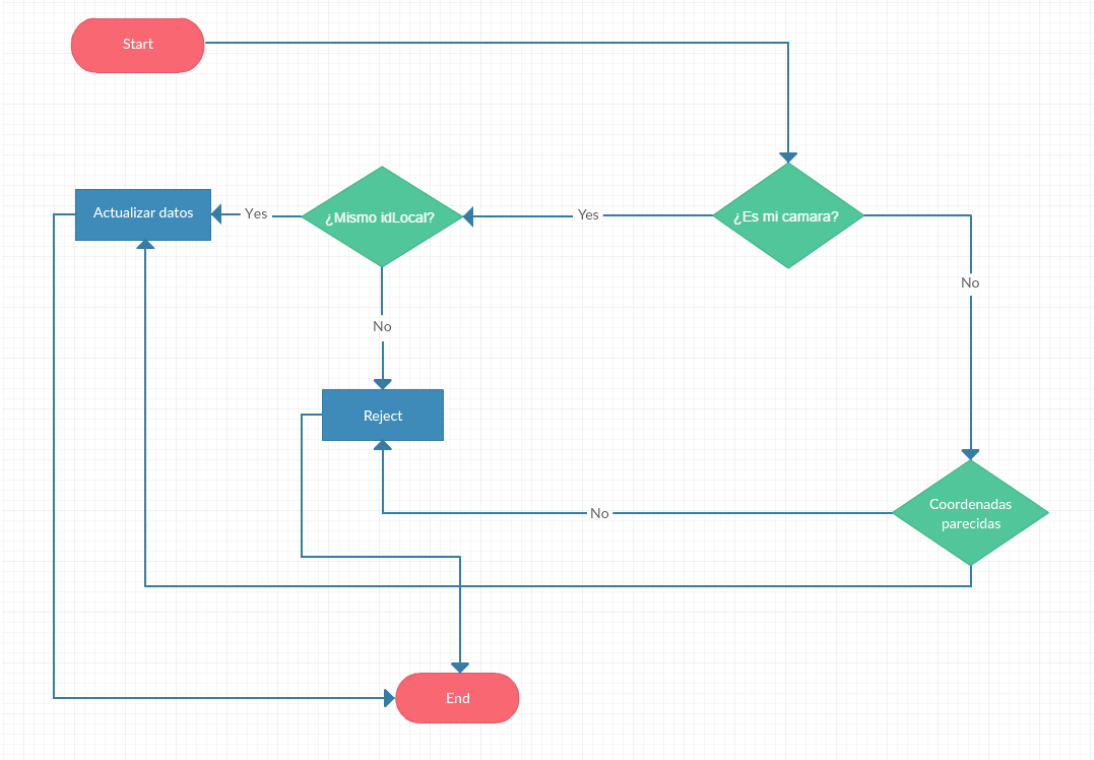

Ilustración 11. Diagrama de flujo tras las primeras pruebas.

4.4.4 Estimación de posiciones

Para realizar las estimaciones de posiciones estamos haciendo uso de un filtro de Kalman, se han explicado sus rasgos generales en 3.7.1 de este documento. La predicción de movimiento vamos a modelar cuatro variables dentro del filtro de Kalman, la primera hará referencia a la posición en el eje X, la segunda hace referencia a la posición en el eje Y, también se modelaran las velocidades en el eje X e Y. De esta forma si se da la situación de que no se pueda actualizar el filtro con valores reales leídos de las cámaras, se generará una posición estimada en base a la anterior posición, y a la última velocidad conocida.

Esta estimación será llevada a cabo con la siguiente ecuación:

$$X_{t+1} = M * X_t$$

$$M = \begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$X_t = \begin{matrix} x \\ y \\ V_x \\ V_y \end{matrix}$$

Para ello vamos a utilizar una librería con la que podremos realizar esto, y que posteriormente se realizará la fase de corrección del filtro, la librería en cuestión se llama Pykalman[17]. Para ello se han realizado pruebas para comprobar que dicho filtro fuera lo más fiable posible para nuestro problema, debe ser capaz de soportar sin desviándose lo menos posible de la realidad en el caso de que falte una cantidad considerable de los datos a leer, en este caso lecturas de posición y velocidades.

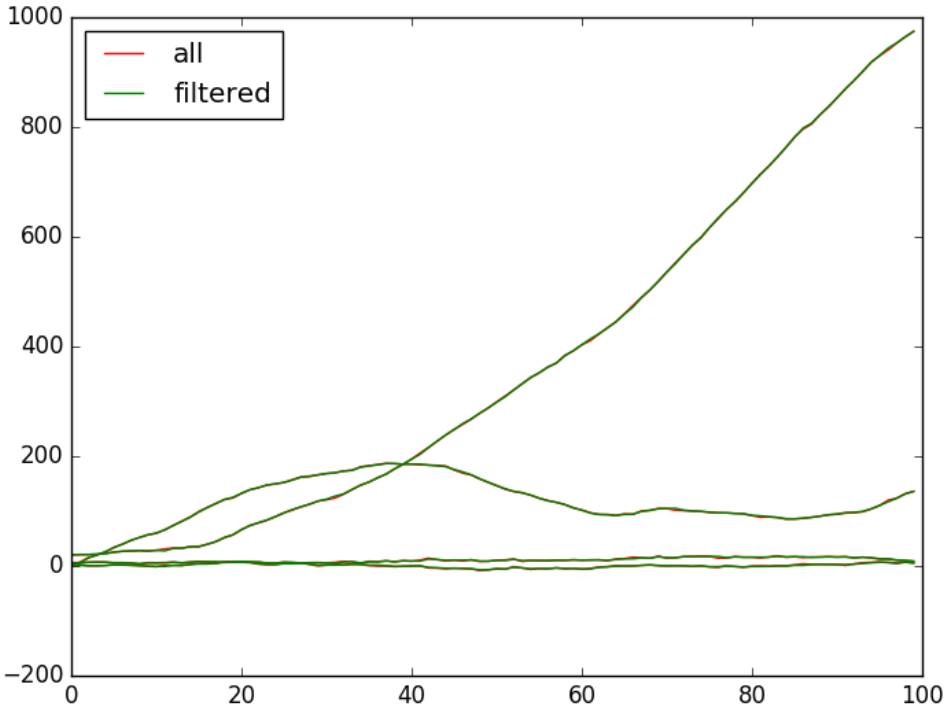

Ilustración 12. Prueba funcionamiento Pykalman perdiendo 1 / 5 de los datos

En Ilustración 12 podemos ver el comportamiento del filtro, para una prueba en la que perdía una de cada cinco mediciones, en este caso un conjunto de números generado. Como se puede ver, prácticamente no hay diferencia entre los valores reales (línea roja) y los valores que nos proporciona el filtro (línea verde), se puede decir que se encuentran prácticamente superpuestas durante todos los valores. Se podría pensar que como solo está perdiendo un valor de cada cinco el valor generado

con el filtro aunque no fuese muy bueno, no influiría en gran medida, por eso también se realizó la misma prueba en el caso de que se perdieran cuatro de cada cinco valores.

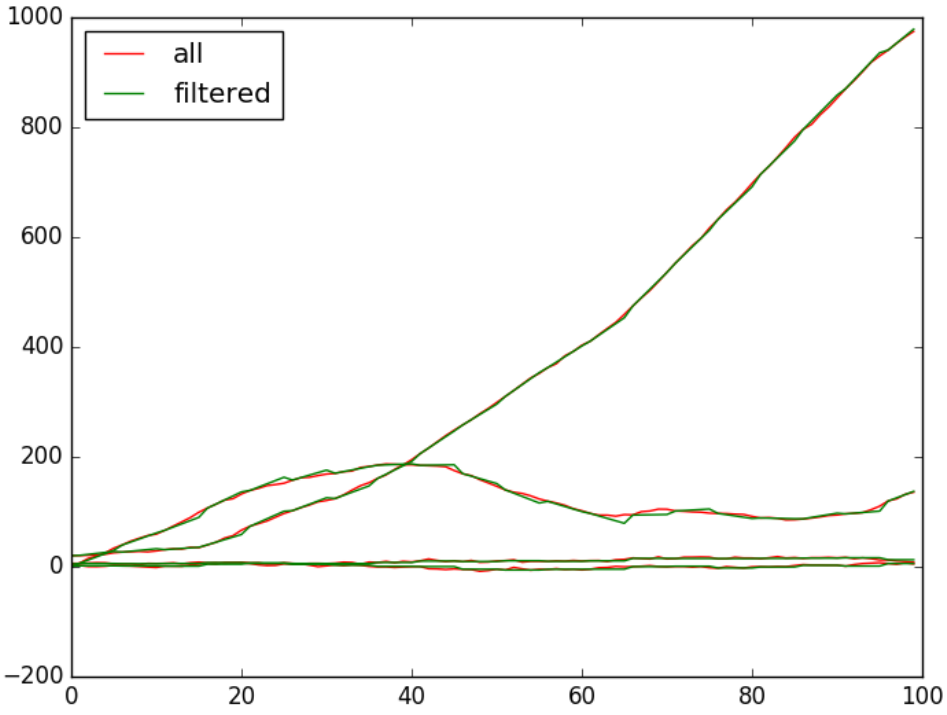

Ilustración 13. Prueba funcionamiento Pykalman perdiendo 4 / 5 de los datos

Se puede observar en Ilustración 13 como a pesar de que ya no se solapan ambas líneas en prácticamente su completo recorrido, se asemejan muchísimo, por lo que podemos asumir que el método de generación de posiciones para cuando no tengamos lectura de un individuo es bastante fiable.

A continuación se mostrará el diagrama de flujo del algoritmo que se encarga de actualizar el filtro de Kalman de nuestro individuo.

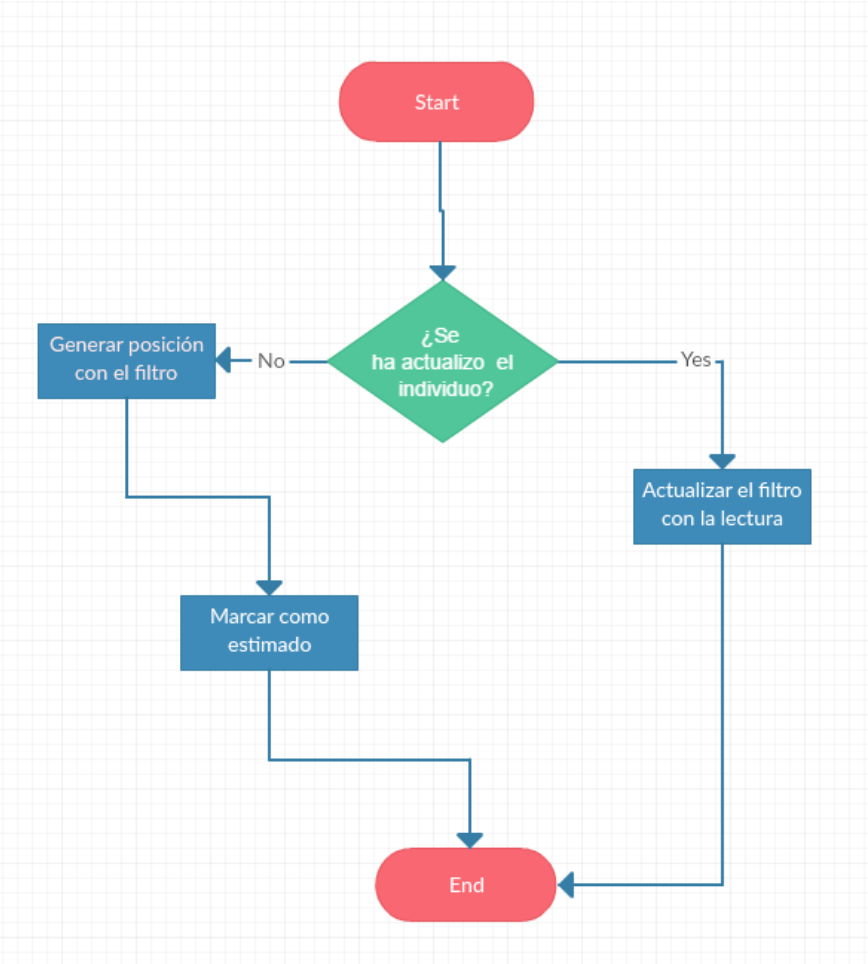

Ilustración 14. Diagrama de flujo updateKalman.

Como se puede ver en la ilustración anterior es un diagrama muy simple en el que se actualiza el valor del filtro en el caso de que se haya leído un valor en esta ejecución, o se genera una posición con el filtro para ese individuo y se marca como posición estimada. Con esta sencilla técnica podemos disponer de una posición para el individuo aunque no dispongamos de la posición, aunque tiene el problema de que para generar dichos valores estaremos generando los valores con los últimos valores de velocidad leídos.

4.4.5 Condiciones para eliminar un individuo del sistema

Aquí se va a explicar lo necesario cuales son las especificaciones del método que es el encargado de eliminar a un individuo de nuestro sistema, como es costumbre este es el diagrama de flujo de dicho algoritmo.

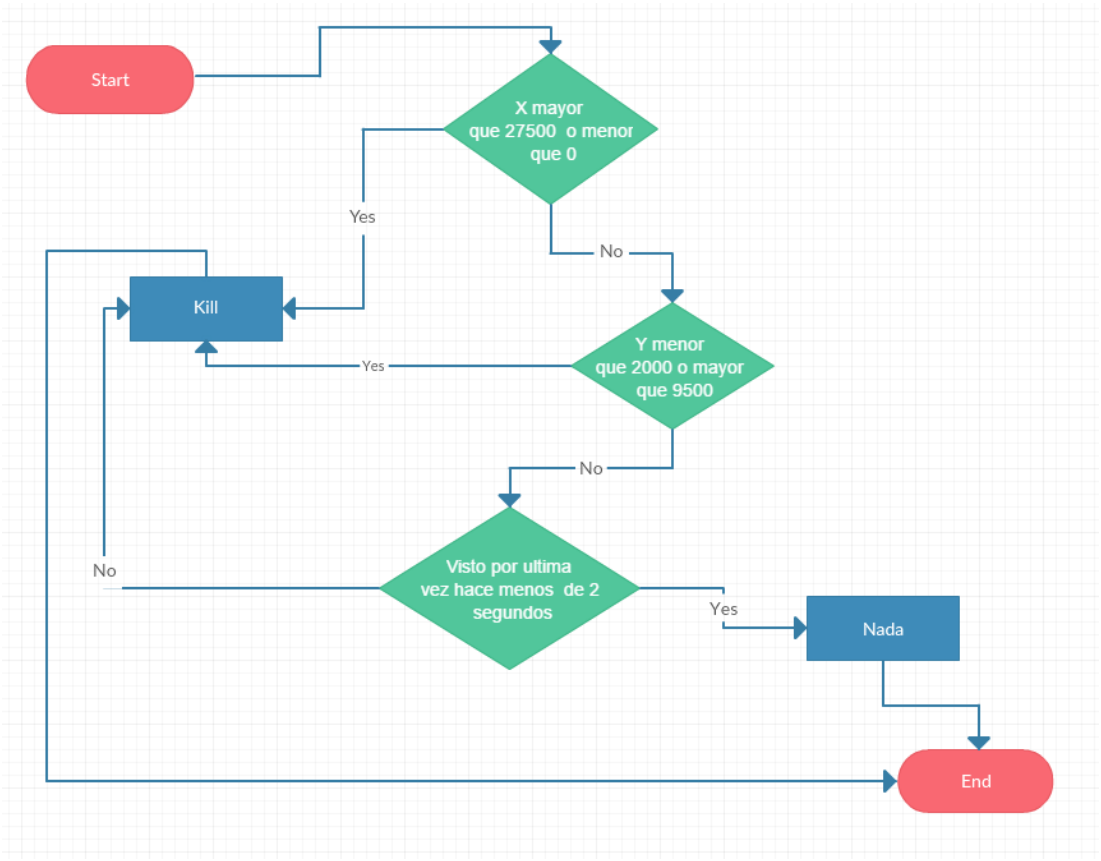

Ilustración 15. Diagrama de flujo del metodo de borrado de individuos.

Se puede ver que es un método bastante simples que comprueba que la posición del individuo no se haya salido de los límites, este caso solo se puede dar si el filtro va generando nuevas posiciones ya que el individuo no está en el campo de visión de las cámaras. También se comprueba que se haya tenido lectura de dicho individuo de una cámara en un tiempo menor de dos segundos, para así evitar el problema de que se quedaran individuos en el sistema al perder la lectura de las cámaras.

4.5 Pruebas y problemas

Tras realizar la implementación de esta fase del desarrollo nos encontramos con que pese a generar bien y poder seguir bien la ruta de una persona aparece un gran problema, la aparición de individuos debido a las lecturas realizadas por varias cámaras simultáneamente que no hemos tenido en cuenta en ningún momento durante la realización del proyecto. Por ello tendremos que plantear una solución a

dicho problema pero que en ningún momento perjudique el funcionamiento del sistema.

4.6 Solución al problema

Para solucionar este problema se optó por una solución en apariencia trivial, pero que nos permitía eliminar a dos personas de la nueva lista de personas, alterando el diagrama de flujo de nuestro método de actualización quedando de la siguiente manera.

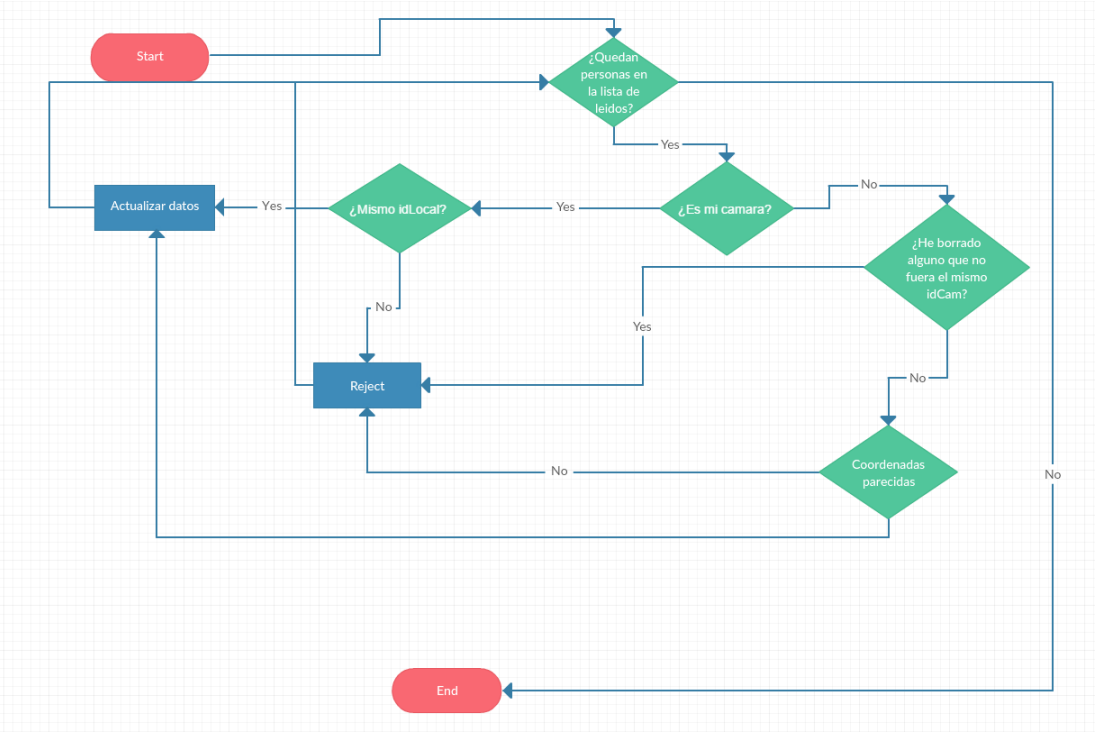

Ilustración 16. Diagrama de flujo modificado para borrar dos personas.

Gracias a esta modificación seremos capaces de eliminar un elemento desde la misma cámara en la que estaba nuestro individuo, siempre y cuando tengan el mismo idLocal, por lo tanto sean la misma persona, y podremos borrar una persona extra de la lista de personas leídas, esto se lo conseguiremos teniendo una variable local en el método que nos permita saber cuándo hemos borrado un elemento que no estuviera en la misma cámara que nuestro individuo, para así realizar ese borrado solamente una vez.

4.7 Problemas encontrados con esta solución

Tras probar durante una semana el proyecto con la solución actual y tras encontrar infinidad de resultados distintos, se optó por realizar una traza escribiendo los valores leídos de las cámaras para ver cuál era el problema, pues seguían apareciendo personas en el sistema, aunque se había solucionado el problema de las zonas solapadas en las que dos cámaras podían detectarnos. También se descubrió que ahora aparecían personas con mucha menor frecuencia, por lo que se investigó por ahí. Se llegó a la conclusión que debido a una mala colocación de las cámaras, que hubieran sido movidas, o alguna otra incidencia se estaban teniendo medidas solapadas de tres cámaras a la vez. Viendo que este problema no tenía otra solución que trabajar en el software de tracking hasta mejorarlo para que no tengamos este problema, hay que buscar otra aproximación a la solución.

4.8 Solución al problema

Para solucionar este problema, se planteó que la solución debería ser la de ser capaz de eliminar todas las apariciones que sean la misma persona en la misma llamada al método de actualización. Para ello se plantearon varias ideas como pueden ser la eliminación de la restricción de borrar solo a dos personas (una en la cámara local y otra en otra cámara), pero esta idea se descartó debido a que en el caso de que dos personas fueran caminando juntos, se convertirían en una única persona en el momento en el que cambiaran de cámara, se planteó la posibilidad que al final se tomó como opción definitiva, la cual consistía en poder borrar una persona de cada cámara, pese a no ser una buena opción pues en el cambio de cámara también se podía perder información de personas, eso se solventó parcialmente incrementando el umbral de distancia a la que considera el sistema que es la misma persona. Tras decidir implementar esta decisión queda de la siguiente manera el diagrama de flujo del método actualizar.

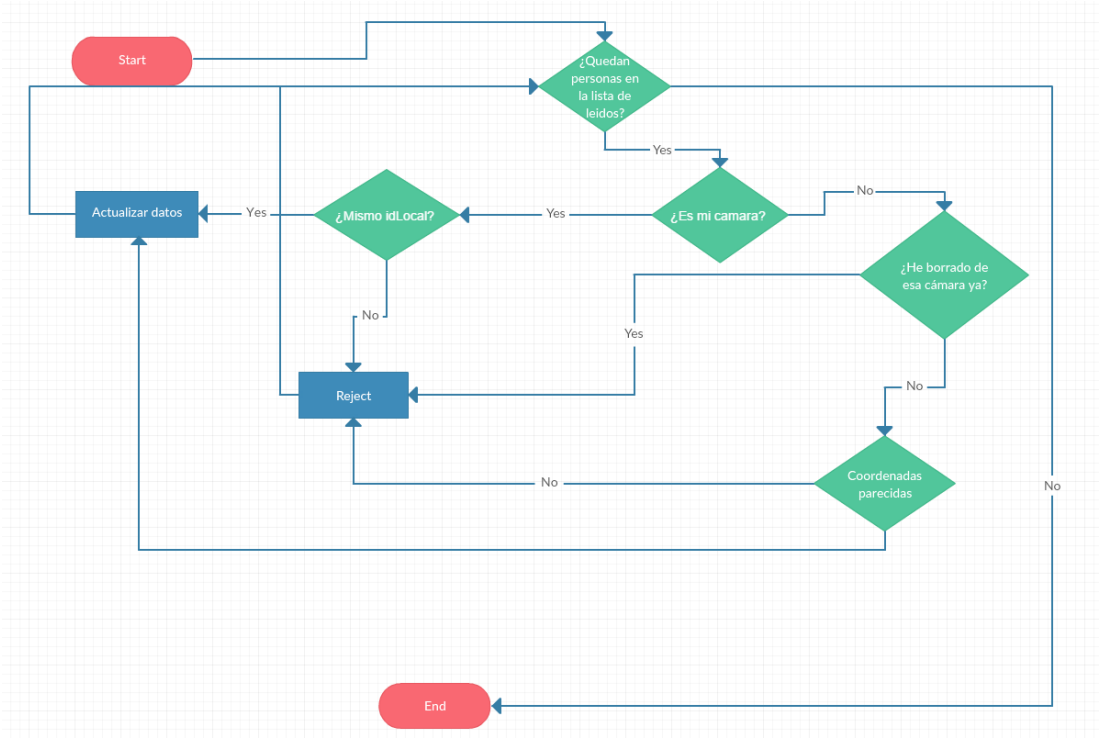

Ilustración 17. Diagrama de flujo con borrado de cada cámara.

Para realizar esta modificación se ha introducido en el algoritmo una máscara que se encargará de llevar la cuenta de si hemos borrado en esa cámara, de tal forma que solo habrá que consultar la posición del vector que hace referencia al idCam de la persona con la que estamos comparando y ver si todavía podemos borrarlo.

4.9 Problemas encontrados con esta solución

Con esta aproximación se mitigaron enormemente los problemas que derivaban de la generación de personas de más en el sistema, pero apareció un problema con el que no se contaba, nuestro sistema en este momento está eliminando a personas que no debe en el intercambio de cámaras y en el momento de pasar a través de una zona a oscuras del sistema. Además del problema comentado anteriormente, también se nos sigue presentando el problema de siempre, y es que en ciertos momentos aparecen puntualmente una persona en un ciclo de lectura, el objetivo de la siguiente fase de la solución será el de minimizar al máximo posible el caso de apariciones de personas en la zona intermedia del sistema y a su vez modificar la configuración de forma que no sea tan agresivo a la hora de eliminar individuos del sistema.

4.10 Solución al problema y estado final del sistema

Visto que no podemos solucionar nuestro problema por el camino de modificar el método de actualizar el individuo, se ha buscado otra aproximación para eliminar los problemas contados en 4.9. Esta solución pasará por realizar un filtrado de los datos antes de generar un individuo nuevo en el sistema, por lo tanto no modificaremos la clase individuo nada más que para proveernos de un método para obtener el número de ocurrencias que ha tenido un individuo en el sistema, y para añadir dicha variable que lleve la cuenta. Una vez hecho esto modificaremos el "specificworker.py" para añadir el filtrado. Modificando el diagrama de flujo ya visto en Ilustración 8 quedándonos el siguiente.

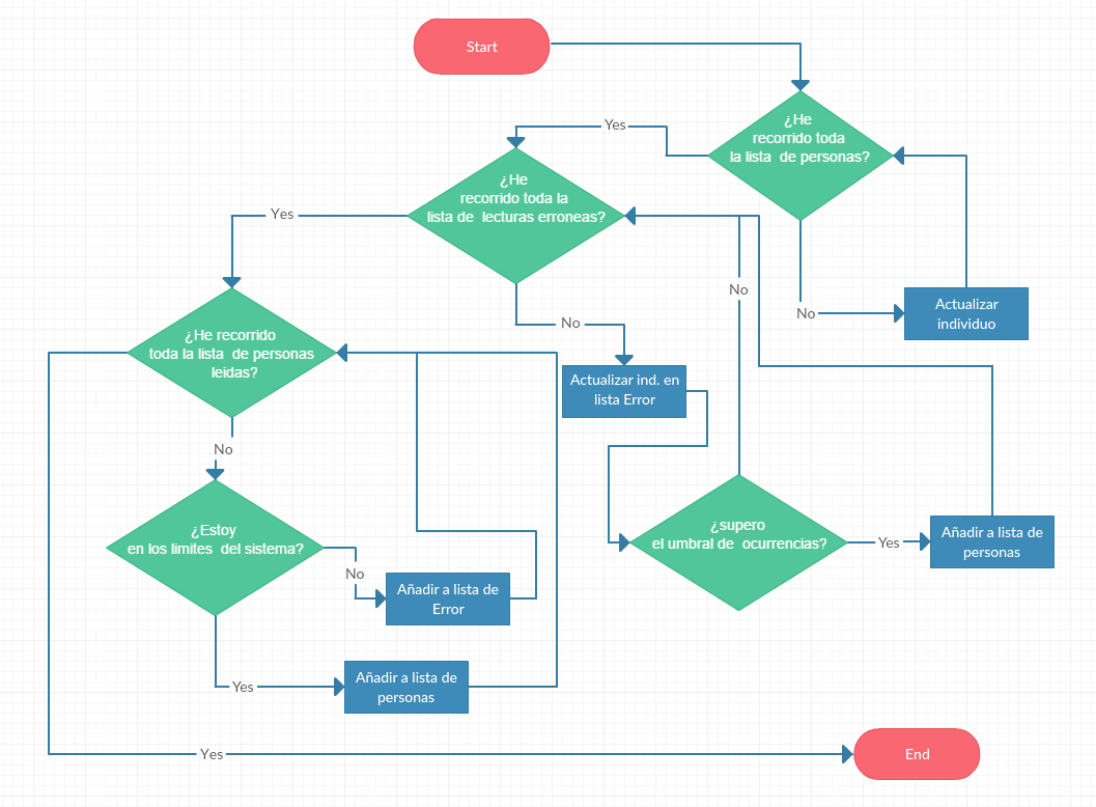

Ilustración 18. Diagrama de flujo final del componente hallMonitor.

En este diagrama podemos ver como el componente hallMonitor se ha complicado un poco desde la última vez que vimos su diagrama de flujo, ahora se ha implementado una lista adicional que hará de buffer para insertar ahí todas las lecturas que no correspondan con una lectura de persona entrando al pasillo, ya sea

por cualquiera de los lados, de forma que primero pasan a esa lista que actúa como filtro, y en el caso de no volver a tener lectura de ese individuo puede eliminarse del sistema pues es una lectura anómala puntual, en el caso de que sea una lectura real, al superar el umbral de ocurrencias que tenemos fijado en el sistema será traspasado a la lista de personas reales del sistema. Con esta implementación y con las configuraciones posibles a través del fichero de configuración para activar o desactivar el método de borrado desde cada cámara del sistema, en el caso de que no sea necesario.

También hay que mencionar, aunque se explicará en detalle en el apartado 5, debido al tipo de sensor con el que las cámaras captan la profundidad, para que el sistema funcione correctamente con indiferencia de la hora o estado del día se ha tenido que utilizar unos toldos para crear un entorno con una luminosidad estable.

4.11 Script de visualización

Para la realización de las pruebas hay algo de lo que hemos hecho uso pero que todavía no se ha discutido a lo largo de esta documentación, eso es el script de visualización de los resultados provistos por el componente hallMonitor. Para ello esta vez no usaremos un componente creado con Robocomp, usaremos la librería PySide[18] para generar las interfaces gráficas con Qt. Se nos presenta el problema de generar todos los ficheros fuentes de la interfaz de comunicación de ZeroC para poder usarlo en dicho script, ya que Robocomp hace esa parte por nosotros.

4.11.1 Generación del código fuente

Para comenzar debemos generar todos los ficheros Python que se encargarán de la configuración para eso nos crearemos una carpeta donde copiaremos la interfaz ICE que usaremos para comunicarnos en este caso “readHall.ice”. Una vez tengamos esto hecho usaremos el siguiente comando para generar todos los ficheros necesarios:

```
slice2py readHall.ice
```

Una vez generado estos ficheros solamente tendremos que realizar el siguiente import desde el script de lectura:

```
import readHallComp
```

Con esto podremos utilizar la interfaz ICE desde nuestro script de visionado, lo siguiente representa el diagrama de flujo de nuestro script de visionado.

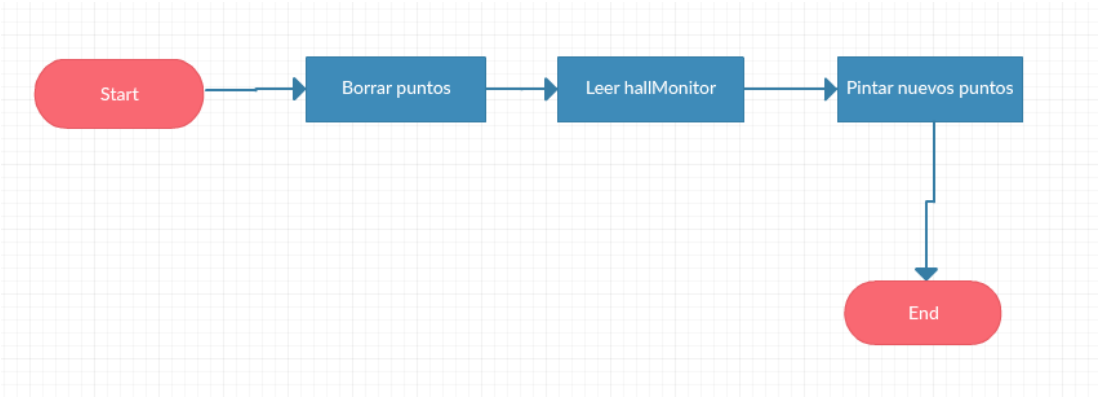

Ilustración 19.Diagrama de flujo readMonitor.

Con esto queda concluida la sección dedicada a la implementación y desarrollo del proyecto, cuyo código puede encontrarse integro en mi repositorio de GitHub[19].

5 RESULTADOS Y DISCUSIÓN

En esta sección se discutirán los resultados finales obtenidos del sistema, además de los problemas que no se han podido solventar o se han solventado solo parcialmente, para comentar los resultados finales se han realizado videos tanto de la salida proporcionada por el visualizador, como de la situación real del sistema para que podamos ver en contraste como funciona.

5.1 Seguimiento de una persona

Para mostrar el resultado en el seguimiento de una única persona en el pasillo se ha grabado el siguiente video enlace al video.

En este video podemos ver cómo podemos seguir perfectamente una ruta en línea recta, pero tarda un poco en aparecer la primera vez debido a la implementación del segundo buffer para evitar los duplicados. También se puede ver que prácticamente no existe problema a la hora de realizar el seguimiento a la vuelta, cuando vuelve haciendo una trayectoria que no era una línea recta, produciéndose un salto hacia atrás en la visualización debido al punto que se seleccionó a la hora de actualizar cuando había dos puntos en el sistema entre los que elegir, ambos de cámaras distintas.

5.2 Seguimiento de dos personas

En otra prueba se planteó el paso de dos personas a la vez, y realizando el mismo tipo de grabación que en el apartado anterior, el video lo podemos encontrar aquí.

Se puede ver cómo avanzan separados por una distancia de varios metros para no generar problemas de oclusión de los que se hablará más adelante. En este video podemos ver como igual que en el video anterior se tarda en visualizar a los sujetos de pruebas, en el momento de la ida el sistema funciona perfectamente sin fallo, y en el momento en el que la primera persona se gira y empieza a volver, mientras la segunda todavía no ha llegado el cruce se produce correctamente en nuestro sistema, pero sí que llega a existir un problema en el momento de la vuelta ya que se pierde del sistema momentáneamente a la primera persona que está volviendo pero es recuperada en las siguientes lecturas de las cámaras.

Esta prueba no solo se limitó a realizarse en la situación más favorable para el sistema, también fue realizada con un caso muy perjudicial, que ya se ha nombrado anteriormente, el cual es la oclusión, dicho video puede encontrarse aquí.

Como se puede apreciar claramente en el video en el cambio entre las cámaras tanto en la ida como en la vuelta, se produce un problema, y es que al usar los datos de la cámara de la pared de enfrente, recordemos el mapa de cámaras explicado en Ilustración 2, se produce un efecto de oclusión ya que la persona que en las imágenes se ve a la izquierda es más grande que la persona que se encuentra a la derecha, por lo que no permite que la vea la cámara, ya que además están avanzando a la vez. Hay que destacar que una vez vuelven a entrar en una cámara de la zona de la

derecha se vuelve a tener constancia de la segunda persona, y se la vuelve a introducir al sistema con un identificador nuevo.

También hay que destacar que puede existir una pequeña desincronización entre la imagen real y la imagen del visualizador, debido a que el video ha tenido que ser montado tras grabar ambas partes con dispositivos distintos, ya que durante la elaboración de todo el proyecto, no se nos ha provisto de acceso a las imágenes proporcionadas por las cámaras por parte de la empresa, y el soporte al que fui remitido tampoco hizo nada por ayudar a obtener las imágenes.

5.3 Problemas y alternativas

En esta subsección se explicarán los problemas que todavía quedan en el sistema y se citarán algunas alternativas que podrían realizar la misma tarea de una forma distinta, o de alternativas posibles para solucionar los problemas restantes del sistema.

En primer lugar y como se dijo anteriormente, al deberse a unas cámaras que captan la profundidad con un sensor infrarrojo, se nos presenta un problema muy grave debido a la localización en la que está presente nuestro sistema a mediados del proceso de desarrollo del proyecto de que el sistema era sensible a la hora del día en el que se realizaban las pruebas, esto hizo pensar en que la radiación solar estaba afectando al sistema, en los momentos en los que más luz entraba en el pabellón desde los tragaluces era totalmente imposible reconocer a nadie que no estuviera resguardado en la pequeña cornisa en la que no daba la luz solar, mientras en los momentos en los que no había luz solar, o la luz solar no era abundante se podía hacer el seguimiento correctamente y aumentaba la distancia de visión de las cámaras. Este problema es debido a que la luz solar también se encuentra formada por componentes infrarrojas, por lo que la mejor solución que se pudo utilizar para este problema es cubrir el tragaluz en la zona estudiada con unos toldos. Una vez realizado este parche al sistema, el rendimiento del sistema mejoró una cantidad considerable, además de proporcionarnos unas condiciones estables para desarrollar y solucionar los problemas.

Otro problema que queda en el sistema, es debido al funcionamiento del sistema, como se pudo ver en los videos anteriores, la oclusión entre individuos es un

problema que no se puede eliminar, ya que se debe al propio funcionamiento de las cámaras, a las cuales hay que destacar que no hemos tenido acceso, por lo que la única forma de poder mitigarlo un poco sería con acceso a las imágenes RGB de las cámaras.

Otro problema también presente es la desaparición de individuos del sistemas como puede verse en un video, con su reaparición un segundo después, esto no se ha conseguido averiguar a que es debido, pero quizás pueda ser arreglado ajustando los parámetros del sistema y con un conjunto de pruebas intensivo.

También se tiene presente la alternativa de utilizar un servidor comunicado mediante una interfaz ICE que preste los servicios de YOLO[20], que ya ha sido desarrollado y está en estado funcional, como un apoyo a la hora de eliminar falsos positivos y evitar el borrado de individuos cuando no sea necesario.

6 CONCLUSIONES

En esta sección se van a explicar las conclusiones obtenidas tanto en lo referente al proyecto, como las conclusiones obtenidas por mí a lo largo del desarrollo del proyecto sobre las tecnologías utilizadas y por último se citaran algunas posibles líneas para seguir el proyecto.

Las conclusiones que se pueden obtener del proyecto son que se ha conseguido implementar un sistema funcional, con ligeros problemas y sensible a la luz solar, pero que es capaz de realizar el seguimiento de trayectorias de personas dentro de la zona sensorizada para ello, aunque no se ha conseguido tener el objetivo inicial, que era que nuestro sistema fuera capaz de mantener un seguimiento de todas las personas presentes en el espacio a estudiar, esto no ha podido llevarse a cabo como se ha explicado anteriormente, no hemos podido acceder a las imágenes de las cámaras para poder apoyarnos en ellas y solucionar algunos problemas, el problema de la luz solar solo se ha podido mitigar con un parche que han sido los toldos para evitar la incidencia de la luz solar, no se ha tenido mucha facilidad por parte de la empresa que instaló las cámaras, retrasando durante algo más de un mes el propio inicio del proyecto solo por no enviarnos la interfaz que utilizaban las cámaras y sin la cual no podíamos ni siquiera leer la información de sus cámaras, tanto con el soporte que han dado para la lectura de las imágenes, las cuales a día de hoy todavía

no tenemos respuesta, todo esto consecuentemente forzando a que la lectura/presentación del proyecto se realice una convocatoria más tarde de lo previsto.

En lo referente a las conclusiones personales he aprendido cosas que desconocía totalmente de Python, tanto de los métodos para factorías en clases, como de decoradores, como de la implementación de la comunicación utilizando el framework ICE, como de como gestiona la memoria dicho lenguaje a la hora de generar iteradores sobre colecciones. He aprendido gran cantidad de detalles del funcionamiento del framework Robocomp, del código ya generado por el generador de código principalmente, del framework ZeroC ICE, también he aprendido como funciona por detrás traduciendo el código de las interfaces en código para realizar las implementaciones, he aprendido también varios trucos que desconocía de Git.

Cabe destacar también después de todo el trabajo realizado que quedan ciertas vías para continuar el proyecto, las cuales personalmente creo que son las siguientes:

- Implementación de un visualizador online con el mapa 3D del pasillo del pabellón, generando avatares en 3D donde nuestro sistema nos dijera que existe una persona utilizando la siguiente herramienta ODG.JS[21], se podría generar la comunicación hacia nuestro componente ya que ZeroC da soporte para javascript.
- Adicción de las imágenes RGB de las cámaras en el sistema ya realizado, de tal forma que se pudiera eliminar los casos de falsos positivos e intentar eliminar de esta forma las confusiones descritas en el apartado de problemas.
- Utilización de cámaras alternativas, trabajando sobre un servidor de YOLO debido a que no tenemos acceso a las imágenes propias de nuestras cámaras.
- Aumento del espacio de trabajo a todo el pabellón, ya sea con esta tecnología como con otras alternativas, como pueden ser cámaras IP, conectadas a un componente que se comunique con el servidor de YOLO citado anteriormente.

Estas son las vías más inmediatas para continuar el proyecto, variando entre ellas de gran manera, puesto que algunas son relativamente sencillas y otras conllevan un coste considerable.

REFERENCIAS BIBLIOGRÁFICAS

1. *Jderobot.* Available from:<http://jderobot.org/Main_Page>.

2. *Robocomp.* Available from:<https://robolab.unex.es/index.php/robocomp/>.

3. *Zeroc Ice.* Available from:<https://zeroc.com/>.

4. *Smartpolitech.* Available from:<http://smartpolitech.unex.es/index.php/2017/04/20/mission-and-vision/>.

5. *Primesense.* Available from:<https://en.wikipedia.org/wiki/PrimeSense>.

6. *Stereolabs Zed.* Available from:<https://www.stereolabs.com/zed/specs/>.

7. *Patente Sistema RGBD.* Available from:<https://www.google.com/patents/US20070216894?hl=es&dq=primesense>.

8. *Microsoft Kinect.* Available from:<https://developer.microsoft.com/es-es/windows/kinect/hardware>.

9. *Asus Xtion.* Available from:<https://www.asus.com/3D-Sensor/Xtion_PRO/specifications/>.

10. *Intel RealSense R200.* Available from:<https://software.intel.com/en-us/articles/realsense-r200-camera>.

11. *Carnegie Robotics MultiSense S7.* Available from:<http://files.carnegierobotics.com/products/MultiSense_S7/MultiSense_Stereo_brochure.pdf>.

12. *E-Con Systems Tara Stereo Camera.* Available from:<https://www.e-consystems.com/3D-USB-stereo-camera.asp#key-features>.

13. *Narian SPI.* Available from:<https://nerian.com/products/sp1-stereo-vision/>.

14. *Python.* Available from:<https://www.python.org/>.

15. *Scrum.* Available from:<https://en.wikipedia.org/wiki/Scrum_(software_development)>.

16. *Ejemplo De Uso De Robocompdsl.* Available from:<https://github.com/robocomp/robocomp/blob/master/doc/robocompdsl.md>.

17. *Pykalman.* Available from:<https://pykalman.github.io/>.

18. *Pyside.* Available from:<https://wiki.qt.io/PySide>.

19. *Repositorio Del Proyecto.* Available from:<https://github.com/JuanPTM/Hal9000>.

20. *Yolo.* Available from:<https://pjreddie.com/darknet/yolo/>.

21. *Osgjs.* Available from:<http://osgjs.org/>.

22. FARAGHER, Ramsey. *Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation.*

23. J. Porubän; M. Bacíkováand J. Šťastná. *Motivating Students in Component-Based Programming Courses.* , 2016.

24. T. Eidson; V. Eijkhoutand J. Dongarra. *Improvements in the Efficient Composition of Applications Built using a Component-Based Programming Environment.* , 2004.