Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza
1542

ROBOTICS
UNIVERSIDAD DE ZARAGOZA

Evaluación de plataformas de bajo coste para construir un sistema de vídeo-vigilancia

Jorge Andrés Galindo

Director: Ana Cristina Murillo Arnal

Ingeniería Informática
Computación

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Noviembre 2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

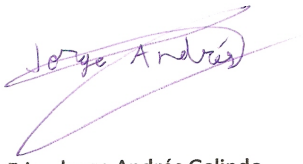
D./Dª. Jorge Andrés Galindo,

con nº de DNI 76924202N en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Evaluación de plataformas de bajo coste para construir un sistema de vídeo-vigilancia,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 23 de Noviembre de 2017



Fdo: Jorge Andrés Galindo

Resumen

En este Trabajo de Fin de Grado se ha realizado la evaluación de diversas plataformas de bajo coste para su futuro uso en proyectos y prácticas relacionados con la robótica y la visión por computador. Para ello se ha utilizado como principal elemento, una Raspberry Pi, con la cual se han probado varias plataformas a evaluar como placas de adaptación para su uso con sensores y motores existentes de LEGO Mindstorms.

Con el objetivo de determinar el alcance del uso de las plataformas anteriormente mencionadas, y como ejemplo de diseño de un prototipo para una tarea real, se ha diseñado, implementado y evaluado un sistema de vídeo vigilancia y monitorización con la Raspberry Pi y las plataformas evaluadas.

Los objetivos del trabajo se han cubierto satisfactoriamente y se pueden agrupar en tres bloques:

Se ha realizado una evaluación y puesta en marcha de todas las plataformas y sensores disponibles, y se han seleccionado los componentes más adecuados. Por un lado se realizo la evaluación de 2 placas para el control de sensores y motores, PiStorms y Dexter BrickPi. En ambas placas se pudieron utilizar los sensores disponibles sin problemas, eligiendo finalmente Dexter BrickPi por su mayor autonomía y robustez. Por otro lado se evaluaron 3 cámaras diferentes, NXTCAMv4, Asus Xtion y Raspberry Pi Camera para su uso con las demás plataformas disponibles. Tras descartar NXTCAMv4 por no estar soportada por las placas, se eligió la Raspberry Pi Camera por su menor consumo y mayor velocidad en comparación con Asus Xtion.

El sistema de *tracking* del prototipo está desarrollado con la biblioteca para visión por computador OpenCV, que permite una instalación sencilla en la plataforma y facilita el procesado y captura de los datos. Se ha realizado una evaluación de los distintos algoritmos de seguimiento disponibles en OpenCV, utilizando datos públicos con datos precisos etiquetados sobre las personas que aparecen en los vídeos del mismo. Esta evaluación ha permitido comprobar el funcionamiento de dichos algoritmos y elegir aquellos que mejor rendimiento presentan con las plataformas elegidas.

Por ultimo se ha construido un prototipo demostrativo con las plataformas *hardware* y los algoritmos elegidos que realizase tareas de vídeo vigilancia y monitorización. El sistema implementado se divide en varias fases:

En primer lugar aplica un detector de personas sobre todos los *frames* que recibe; una vez que encuentra a una persona, comenzará a hacer *tracking* de esa persona; en función de donde indique el *tracking* que se encuentra esa persona, el sistema moverá un motor para orientar la cámara hacía la posición del objetivo; cuando el sistema pierde al objetivo, vuelve a aplicar la detección. Además cada vez que encuentra a una persona y empieza a realizar *tracking*, sube una imagen de dicha persona a Dropbox, permitiendo acceder remotamente a los datos.

El prototipo funciona de forma correcta y es capaz de monitorizar al objetivo en tiempo real, siempre y cuando la velocidad del objeto a seguir no sea muy alta, para lo cual haría falta una capacidad de cómputo mayor.

Índice general

Capítulo 1

Introducción

Este capítulo introduce el proyecto realizado, sobre componentes de prototipado rápido y su posible aplicación a sistemas de monitorización inteligentes (Figura 1.1), explicando la motivación del trabajo y objetivos más concretos.

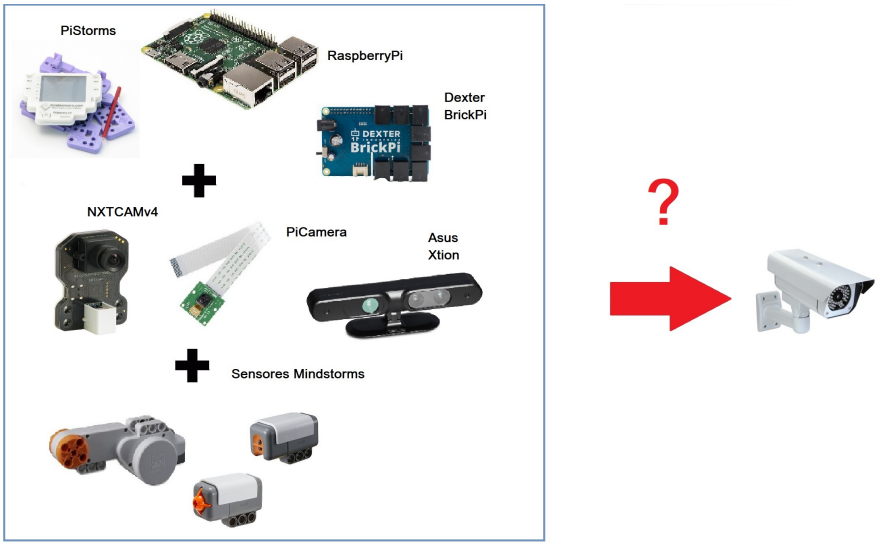
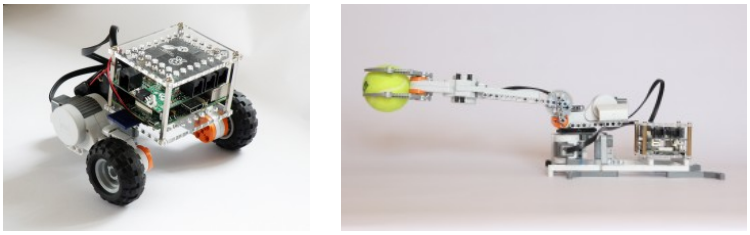


Figura 1.1: El objetivo de este proyecto es evaluar distintos componentes de bajo coste y su posible utilidad para construir sistemas inteligentes, en concreto un sistema de vigilancia.

1.1. Motivación y Contexto

Este trabajo esta motivado por el auge en la variedad y cada vez más alta calidad de los componentes de prototipado rápido. Por un lado, las placas de



(a) (b)

Figura 1.2: (a) Robot de control remoto con Raspberry Pi. (b) Brazo robótico con Raspberry Pi. (Fuente www.dexterindustries.com/BrickPi)

prototipado rápido, como la Raspberry Pi[1] o Arduino[2], cada vez tienen más capacidad de cómputo. Esto hace que aplicaciones más complejas, por ejemplo de visión por computador, se puedan ejecutar en estas plataformas. Por otro lado, también encontramos mayor facilidad para conectar diferentes dispositivos y sensores, tanto directamente en los puertos USB o pines de las placas, como utilizando placas de adaptación para utilizar otros sensores y motores, por ejemplo de LEGO Mindstorms[3]. Podemos encontrar múltiples proyectos de robótica combinando la Raspberry Pi con placas de adaptación y distintos sensores, para construir pequeños robots de control remoto o brazos robóticos como los que se pueden ver en la Figura 1.2.

En este trabajo se estudia la posibilidad de construir un sistema de vídeo vigilancia o monitorización basado en plataformas de bajo coste disponibles en el departamento de informática e ingeniería de sistemas de la Universidad, como prototipo demostrador de posibles materiales a utilizar en prácticas de asignaturas relacionadas.

Este proyecto se ha realizado en colaboración con el grupo de robótica, percepción y tiempo real de la Universidad de Zaragoza, que ha proporcionado los materiales que posteriormente han sido evaluados en el proyecto. Estas evaluaciones, determinarán también el uso de los mismos en futuros proyectos y prácticas relacionados con la visión por computador en plataformas de bajo coste.

Encontramos múltiples proyectos que estudian el uso de estas plataformas para vigilancia y monitorización, tanto en foros de desarrollo[4] como en estudios con objetivos más técnicos sobre el rendimiento de este tipo de sistemas [1] [2]. Los objetivos más concretos de este trabajo se detallan a continuación.

1.2. Objetivos, Tareas y Alcance

Como se ha comentado anteriormente, en este trabajo se propone desarrollar un prototipo de monitorización visual montado sobre una Raspberry Pi, analizando las opciones disponibles y restricciones impuestas por las mismas. Hay dos objetivos principales a alcanzar, detallados a continuación junto con

[1] `https://www.raspberrypi.org`
[2] `https://www.arduino.cc/`
[3] `https://www.lego.com/es-es/mindstorms`
[4] `https://www.pyimagesearch.com/`

las tareas realizadas para alcanzar cada uno de ellos.

Selección Hardware. Evaluación de las distintas opciones de *hardware* disponibles para elegir aquellas más adecuadas para este tipo de plataforma/aplicación. Para lo cual se han realizado las siguientes tareas:

- Evaluación de las placas PiStorms[5] y Dexter BrickPi[6], detalladas más adelante en el Capítulo 2, que permiten a la Raspberry Pi conectar y controlar los sensores LEGO Mindstorms. Puesta en marcha de dichas placas, comprobando y valorando el funcionamiento, autonomía y facilidad de uso y programación de cada una.
- Evaluación del funcionamiento de motores, sonares, sensores de luz y otros sensores de LEGO Mindstorms con las distintas placas. Ambas placas disponibles tienen APIs para acceder y controlar distintos sensores, esta tarea consiste en la evaluación y puesta en marcha de los mismos.
- Evaluación de la compatibilidad de las cámaras NXTCAMv4[7], Asus Xtion[8] y Raspberry Pi Camera[9], detalladas en el Capítulo 2, con la Raspberry Pi. Esta tarea abarca desde la instalación de los drivers necesarios para las mismas en la Raspberry Pi, a realización de capturas con las mismas.
- Evaluación de las cámaras compatibles con el sistema desarrollado. Comprobando su facilidad de uso, su consumo de energía, funcionamiento y rendimiento con la Raspberry Pi.

Diseño del sistema. Estudiar, diseñar e implementar el sistema a utilizar en este hardware. En concreto se quería desarrollar un sistema de análisis de vídeo que pueda realizar tareas de seguimiento y monitorización en la plataforma seleccionada. Para ello se han realizado las siguientes tareas:

- Estudio de los métodos ofrecidos por la biblioteca OpenCV[10].
- Elección de un método de detección de personas estándar que diera resultados aceptables en el prototipo diseñado, en este caso se ha usado HOG (Histogram of Oriented Gradients) [3].
- Análisis y evaluación del tiempo de ejecución y la precisión de los distintos métodos de seguimiento disponibles. Este análisis se realizo debido a que estos métodos son una parte primordial en el buen funcionamiento del prototipo en términos de velocidad, y por lo tanto, era necesario elegir el que mejor funcionase en dicha plataforma.
- Implementación de los módulos necesarios del sistema para la realización de las tareas de vídeo vigilancia y seguimiento.

[5] `http://www.mindsensors.com/content/78-pistorms-lego-interface`
[6] `https://www.dexterindustries.com/brickpi/`
[7] `http://www.mindsensors.com/ev3-and-nxt/14-vision-subsystem-camera-for-nxt-or-ev3-nxtcam-v4`
[8] `https://www.asus.com/es/3D-Sensor/Xtion_PRO/`
[9] `https://www.raspberrypi.org/products/camera-module-v2/`
[10] `https://opencv.org/`

Integración. Construcción de un **prototipo demostrativo** con el sistema diseñado y el hardware seleccionado que realice tareas de vigilancia y monitorización. Para lo cual se han realizado las siguientes tareas:

- Diseño y construcción del prototipo mediante el *hardware* seleccionado, sensores y piezas de LEGO Mindstorms para lograr un sistema funcional.
- Pruebas con los diferentes sensores conectados al prototipo para comprobar su correcto funcionamiento.
- Experimentos, tanto en tiempo real como con vídeos grabados por el prototipo para comprobar su funcionamiento ante situaciones reales.

La Figura 1.3 muestra un resumen del tiempo dedicado a estas tareas. En total a la realización de estas tareas se han dedicado aproximadamente 330 horas, de las cuales aproximadamente 70 horas corresponden a la evaluación del *hardware*, 90 horas a la evaluación de los algoritmos de *tracking* y 60 horas para la implementación del sistema. Por otro lado se han empleado 60 horas para la integración de todo el sistema en el prototipo y los experimentos realizados en dicho prototipo.

1.3. Contenido de la memoria

En el presente documento se va a exponer las evaluaciones realizadas a los distintos elementos que forman parte del sistema, así como los resultados y conclusiones obtenidas siguiendo la siguiente estructura:

- **Capítulo 2.** Se describen las características del distinto *hardware* disponible, una comparación del mismo así como las conclusiones obtenidas y elección de lo que formará finalmente parte del sistema.
- **Capítulo 3.** Se describe y analiza los distintos algoritmos de *tracking* disponibles en OpenCV para posteriormente analizarlos para decidir cual funcionará mejor en el prototipo final.
- **Capítulo 4.** Se muestra el prototipo final del sistema de vídeo-vigilancia y se describen las diferentes pruebas de integración que se han realizado con el mismo.
- **Capítulo 5.** Conclusiones finales y objetivos cumplidos, además de futuros trabajos que ofrece el proyecto.

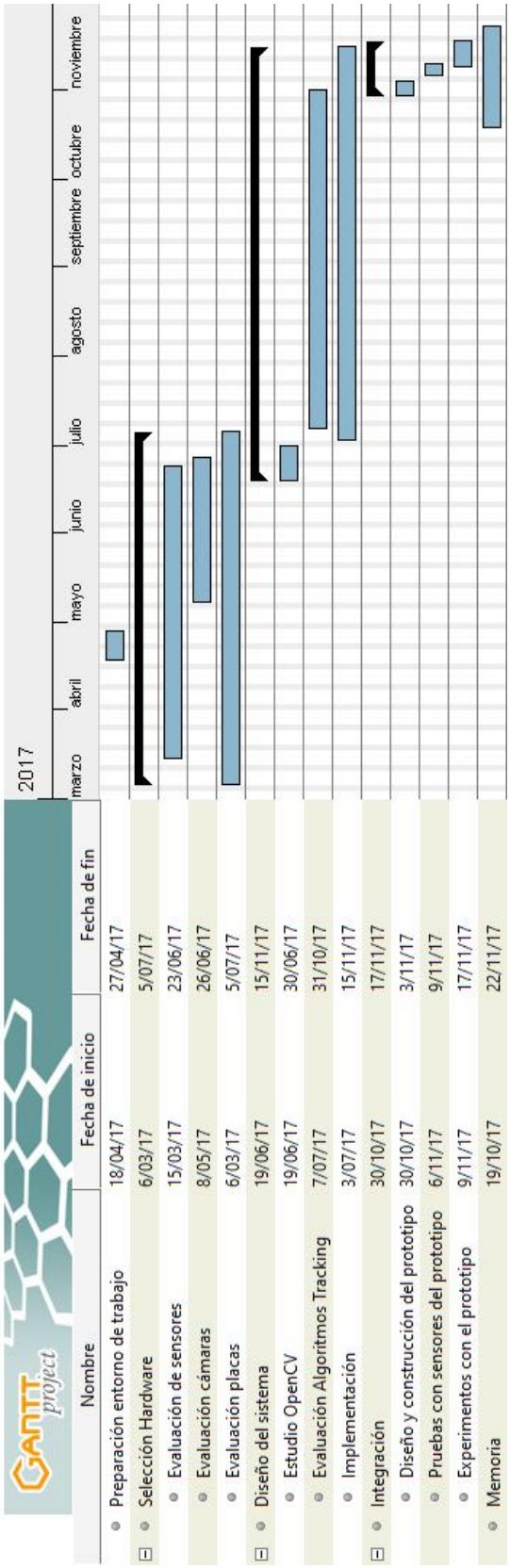


Figura 1.3: Diagrama de Gantt de las tareas realizadas.

Capítulo 2

Plataforma hardware de bajo coste para monitorización

La plataforma *hardware* a utilizar precisará diversos elementos que le permitan realizar las diversas labores de monitorización, como la capacidad de detectar personas y orientar la cámara hacia donde se muevan las mismas.

Los principales elementos de *hardware* a evaluar son la placa de adaptación y la cámara que se conectan a la Raspberry Pi[1] y que junto a esta forman el prototipo de sistema de vigilancia. La placa se encarga de permitir acceder y controlar los diferentes sensores y motores de LEGO Mindstorms[2] disponibles para evaluar en este proyecto. También se evaluará la funcionalidad de estos sensores y motores disponibles. Por otro lado, la cámara captura las imágenes sobre las que se aplican los diferentes algoritmos de visión por computador para realizar labores de vigilancia.

2.1. Requisitos de la plataforma a diseñar

Como se ha mencionado, un objetivo de este trabajo es evaluar la adecuación y posibilidades de las plataformas disponibles en el departamento de robótica para diseñar sistemas inteligentes de bajo coste, de cara a futuros trabajos y prácticas. Por lo tanto, es importante tener esto en cuenta a la hora de evaluar y definir los requisitos del trabajo. Los componentes seleccionados, deberán cumplir varios requisitos que permitan el buen funcionamiento de la plataforma:

- **Facilidad de uso** para su posterior desarrollo y utilización, y facilidad para realizar cambios en caso de ser necesario.
- **Eficiencia**, tanto energética, que permita a la plataforma tener una autonomía aceptable, como de rendimiento, permitiendo a la plataforma realizar sus funciones sin problemas.

[1] `https://www.raspberrypi.org`
[2] `https://www.lego.com/es-es/mindstorms`

2.2. Evaluación de placas de conexión Mindstorms-Raspberry

Las placas de conexión entre los componentes Mindstorms y la Rasbperry Pi disponibles para utilizar en el proyecto son PiStorms[3] y Dexter BrickPi[4], ambas pueden ser observadas en la Figura 2.1. Estas placas, como ya se ha mencionado, se encargarán de acceder y controlar los sensores conectados a la misma plataforma. A continuación se describirán y evaluarán sus principales características para determinar cuál encaja mejor con los requisitos requeridos por el proyecto.

2.2.1. Descripción de las placas consideradas

Esta sección describe los dos modelos de placa que se encontraron y se han considerado en este trabajo.

PiStorms. Las principales características de esta placa son las siguientes:

- 4 puertos para sensores.
- 4 puertos para motores.
- Pantalla táctil, con la que es posible controlar la plataforma sin necesidad de un ordenador.
- Alimentación. Solo permite la alimentación a través de la placa. La plataforma se alimenta con 6 pilas AA. La placa no permite que se conecte la Raspberry Pi directamente a corriente mediante su cargador por lo que solo es posible alimentar la plataforma a través de la placa.

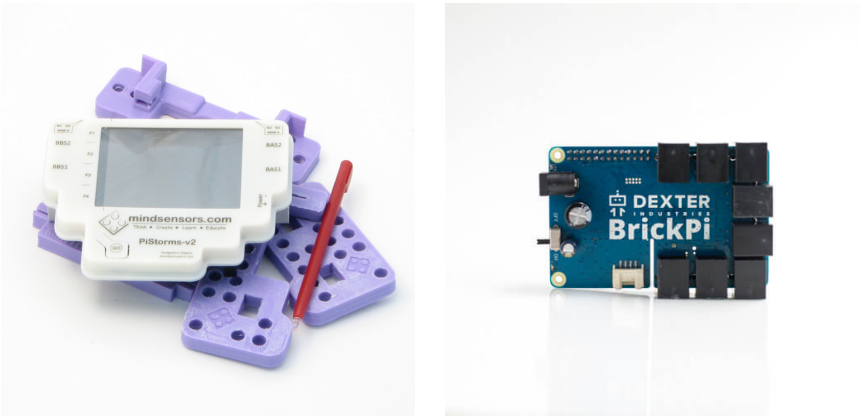
PiStorms puede ser programada mediante los lenguajes scratch y python. Además ofrece una gran cantidad de documentación que facilita su programación, permitiendo manejar los sensores conectados a dicha placa de forma sencilla.

Dexter BrickPi. Las principales características de la placa PiStorms son las siguientes:

- 4 puertos para sensores.
- 4 puertos para el control de motores.
- Control a través de internet.
- Permite alimentación tanto a través de la placa como a través del puerto micro-USB de la Raspberry Pi. La placa se alimenta con 8 pilas AA. Alternativamente la plataforma puede ser alimentada a través de la Raspberry Pi con un cargador, sin necesidad de utilizar pilas o baterías para tal fin.

Entre los lenguajes con los que se puede programar Dexter BrickPi se encuentran scratch, python y java. A pesar de que esta placa no ofrece mucha documentación, programar los sensores y motores conectados a la placa es muy sencillo al ser muy intuitivo.

[3] `http://www.mindsensors.com/content/78-pistorms-lego-interface`
[4] `https://www.dexterindustries.com/brickpi/`



(a) (b)

Figura 2.1: (a) Placa PiStorms. Fuente: `www.mindsensors.com`. (b) Placa Dexter BrickPi. Fuente: `www.modmypi.com`

2.2.2. Evaluación y Discusión

A la hora de comparar ambas placas, se comprobó que PiStorms daba problemas con los cables que conectan su fuente de alimentación a la placa si no se trataba con el suficiente cuidado, mientras que, BrickPi utiliza un conector más sólido lo que lo vuelve más resistente a roturas.

Por otro lado se comprobó que BrickPi cuenta con una mayor autonomía energética que PiStorms. Esto es debido a que PiStorms debe alimentar a una pantalla táctil además de que su fuente de alimentación cuenta con menos pilas que la BrickPi. Por otro lado, BrickPi permite el uso de un cargador lo cual es muy útil cuando se llevan a cabo labores de desarrollo ya que no es necesario vigilar las pilas o baterías.

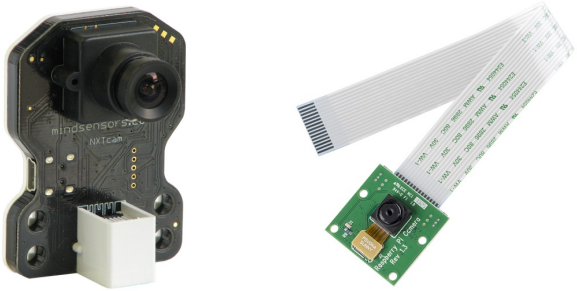
La principal desventaja de BrickPi frente a PiStorms es que cuenta con menos autonomía a la hora de controlarla al no contar con una pantalla como la de PiStorms. Esto obliga a que la plataforma este siempre conectada internet para poder manejarla en todo momento.

Finalmente se decidió utilizar BrickPi principalmente por su autonomía energética, y por su mayor robustez a la hora de construir el prototipo (tanto el sistema de alimentación como las placas quedan sujetos de manera más fija, lo cual da una plataforma más funcional para el tipo de aplicaciones que se quieren realizar en ella). Se pueden encontrar más detalles sobre el uso de Dexter BrickPi en el Anexo A.

2.3. Evaluación de cámaras para el seguimiento visual

Para la realización de tareas de visión en la plataforma, se disponía de 3 modelos de cámaras diferentes: NXTCAMv4, Asus Xtion y Raspberry Pi Camera, que pueden ser observadas en la Figura 2.2. En esta sección se describirán y evaluarán sus principales características.

2.3.1. Cámaras evaluadas



(a) (b)

(c)

Figura 2.2: (a) Cámara NXTCAMv4 (b) Cámara Raspberry Pi Camera (c) Cámara Asus Xtion

NXTCAMv4. La NXTCAMv4 es una cámara que permite almacenar en 8 registros internos distintos rangos de colores. Estos registros le permiten reconocer *blobs* que se encuentren dentro de estos rangos, facilitando de esta forma realizar un seguimiento de dichos *blobs*. La cámara es controlada a través de la placa a la cual es conectada. Sin embargo en el caso de BrickPi esto no era posible ya que la placa no cuenta con funciones que permitan acceder a la misma, complicando bastante su uso. En el caso de PiStorms, se posee un módulo que permite controlar la cámara y realizar diversas acciones con ella. Sin embargo, este módulo parece estar incompleto ya que la mayoría de las funciones que contenía no funcionaban. Debido a estos motivos se decidió descartar esta cámara y no utilizarla en el sistema de vigilancia.

Asus Xtion. La Asus Xtion es una cámara de profundidad, es decir, permite saber de forma aproximada a que distancia de la cámara se encuentran los objetos capturados por la misma. Para ello hace uso de un sensor de infrarrojos. La cámara se conecta directamente a la Raspnerry Pi a través de un puerto USB. Para usarla simplemente es necesario instalar el *framework* OpenNI que permite manejar las distintas funciones de la cámara con facilidad.

Raspberry Pi Camera. La Raspberry Pi Camera es una cámara especialmente fabricada para las Raspberry Pi, con ella se pueden hacer todas las funciones básicas de una cámara de forma fácil y rápida. Permite realizar tanto

fotografías como vídeos de diferentes formas utilizando solamente un módulo ya preinstalado en la Raspberry Pi.

2.3.2. Evaluación y Discusión

Como ya se ha mencionado anteriormente, no fue posible hacer funcionar la NXTCAMv4, por lo que se decidió descartarla, dejando la elección de cámara solamente entre la Asus Xtion y la Raspberry Pi Camera.

Al estar la decisión entre la Asus Xtion y la Raspberry Pi Camera, finalmente se optó por elegir para el proyecto la segunda opción. Uno de los motivos que impulsaron esta decisión fue el menor consumo de la Raspberry Pi Camera, este menor consumo permite que la batería de la plataforma tenga una mayor duración. Otro de los motivos fue la velocidad con la que la Raspberry Pi Camera funciona a la hora de tomar imágenes frente a la Asus Xtion, debido a que precisa de menos recursos, lo cual es de gran importancia en este proyecto ya que se precisa de la mayor eficiencia posible al ser un sistema en tiempo real.

Capítulo 3

Algoritmos de Seguimiento Visual o *Tracking*

Para la parte de *tracking* de la plataforma de vigilancia, se probaron los distintos métodos de *tracking* o seguimiento visual disponibles en OpenCV 3.2[1] para determinar cual podría ser utilizado en la plataforma escogida. A continuación se describen los distintos algoritmos disponibles en OpenCV que se han estudiado.

3.1. Descripción de algoritmos de *tracking*

Boosting. Basado en el algoritmo de AdaBoost que consiste en unir varios clasificadores pequeños en un único clasificador que, en conjunto, sera más robusto [4]. Este clasificador, determinará que píxeles de la imagen pertenecen al objeto a seguir y cuales pertenecen al fondo de la imagen. Este clasificador, al contrario que el AdaBoost original, es entrenado continuamente durante su propia ejecución permitiéndole adaptarse continuamente, para ello considerará como ejemplos positivos el área inicial (proporcionada por el usuario) y aquellos que posteriormente clasifique como positivos, y como negativos considerará elementos que se encuentren fuera de dicha área [5]. Cada vez que se proporcione un nuevo *frame*, el clasificador observará los píxeles cercanos a la anterior posición del elemento a seguir y determinará su nueva localización.

MIL (Multiple Instance Learning). Similar al algoritmo anterior, se basa en la idea de que si el *tracker* recibe un mal ejemplo inicial, clasificará peor conforme reciba imágenes. Para evitarlo este algoritmo utiliza bolsas de ejemplos para clasificar en lugar de ejemplos individuales. En este algoritmo se clasifican las bolsas, y solo es necesario que una bolsa tenga un ejemplo positivo para que toda la bolsa sea clasificada como positiva. De esta forma se consigue mejorar la precisión al indicar la posición del objeto a seguir [6].

KCF (Kernelized Correlation Filters). Basado en los dos anteriores, este algoritmo se basa en el hecho de que en el algoritmo MIL varios ejemplos tienen

[1] `https://docs.opencv.org/trunk/d6/d00/tutorial_py_root.html`

regiones solapadas que hacen que se realicen clasificaciones redundantes. Al saber esto, el algoritmo utiliza diversas propiedades matemáticas (transformada de Fourier y matrices circulares) para evitar clasificar repetidamente dichas regiones, aumentando así la velocidad y la precisión del clasificador [7]. Además la versión implementada en OpenCV utiliza también información aportada por el color de la imagen para obtener mejores resultados [8].

MedianFlow. Este algoritmo observa la trayectoria del objeto a seguir adelante y atrás en el tiempo, es decir, realiza *tracking* del objeto desde el primer *frame* hasta el último y a la inversa, desde el último *frame* al primero. Esto permite detectar errores si ambas trayectorias difieren en algún punto y, al corregir dichos errores, conseguir mejores resultados [9].

TLD (Tracking, Learning and Detection). Este algoritmo divide el *tracking* a largo plazo en 3 subtareas diferentes. Por un lado el *tracker* sigue el objetivo *frame* a *frame*. El detector observa todo lo detectado por el *tracker* hasta el momento y lo corrige, en caso necesario, evitando así que se incrementen los errores del *tracker* con el tiempo. Por último, se estiman los errores que podría tener el detector y se "aprende" de ellos actualizando el detector para que no vuelva a cometer dichos errores [10]. En la implementación de OpenCV la parte de *tracking* es llevada a cabo con el algoritmo Medianflow, anteriormente explicado.

GoTurn. Este es el único algoritmo que precisa de entrenamiento previo antes de realizar su función. Debido a esto y a que ya no precisará de ningún entrenamiento cuando este realizando *tracking*, este *tracker* es uno de los más rápidos, pudiendo, según la documentación, llegar a ejecutarse a 100 FPS. Para realizar el *tracking* utiliza una red neuronal cuya salida será la zona aproximada de la posición actual del objeto [11].

CamShift. Este algoritmo en primer lugar aplica el algoritmo meanshift, que encuentra el objeto a seguir buscando el lugar con una mayor densidad teniendo en cuenta el histograma y anterior posición del objeto. Tras esto camshift aplicará funciones matemáticas para redimensionar y reorientar la ventana de búsqueda. Tras lo cual, se volverá a aplicar meanshift. Este proceso se repetirá hasta obtener los resultados requeridos [12].

3.2. Evaluación de los algoritmos

Esta sección presenta los experimentos realizados para evaluar que algoritmo, de los anteriormente presentados (Sección 3.1), resulta más adecuado para el sistema planteado. Así como las conclusiones obtenidas de dichos experimentos.

3.2.1. Entorno de evaluación

Para la evaluación de estos algoritmos se han considerado principalmente su tiempo de ejecución y su eficacia a la hora de realizar la tarea de seguimien-

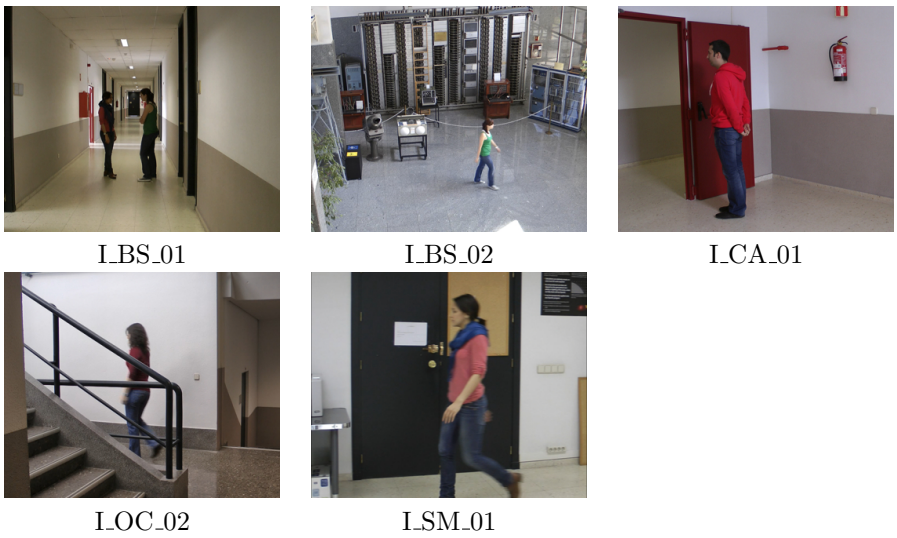


Figura 3.1: Ejemplos de imágenes de cada una de las secuencias elegidas para la evaluación inicial (detalladas en Tabla 3.2)

to/detección. Para una evaluación objetiva, se ha utilizado un *dataset* público, LASIESTA[2], detallado a continuación.

3.2.1.1. Datos utilizados para evaluación: *dataset* LASIESTA

El *dataset* LASIESTA [13] está formado por diversas secuencias, tanto de interior como de exterior, de personas y otros objetos en movimiento. Debido a que este proyecto esta más enfocado a realizar pruebas en entornos de interior se decidió utilizar solo estas secuencias.

Las secuencias están formadas por *frames* con una resolución de 352x288 píxeles. Cada *frame*, a su vez, esta etiquetado mediante segmentación detallada realizada a mano que indica qué píxeles de la imagen pertenecen a cada persona de la escena.(Figura 3.2).

A su vez, las secuencias están divididas en distintos tipos con unas determinadas características descritas en la Tabla 3.1.

Para la evaluación de todos los algoritmos se eligieron 5 secuencias diferentes con las que se obtuvieron los tiempos empleados en procesar cada *frame* y la precisión con la que se realizó el seguimiento. Las secuencias elegidas y sus principales características se muestran en la Tabla 3.2.

3.2.1.2. Métricas utilizadas

Las métricas utilizadas para evaluar los algoritmos estudiados han sido las siguientes:

Tiempo medio de *tracking* (t). Tiempo medio en segundos que el algoritmo ha empleado en los *frames* evaluados para localizar al objetivo. Para su cálculo

[2] `https://www.gti.ssr.upm.es/data/lasiesta_database.html`

Figura 3.2: Ejemplo de datos etiquetados del *dataset* LASIESTA [13].

Tabla 3.1: Tipos de secuencias del *dataset* LASIESTA

Tipo de Secuencia	**Descripción**
SI	Secuencias simples, sin nada especial.
OC	Secuencias con oclusión de objetos en movimiento.
IL	Secuencias con cambios de iluminación.
MB	Secuencias en las que se abandonan objetos o estos son sustraidos del lugar.
BS	Secuencias con objetos en movimiento desde el primer frame.
MC	Secuencias grabadas con cámara no estática.
SM	Secuencias que simulan distintos movimientos de la cámara.

Tabla 3.2: Subconjunto de secuencias de LASIESTA elegidas para evaluación inicial.

Secuencias	Duración (segundos)	*Frames*	*Frames* con personas
I_BS_01	11	275	183
I_BS_02	11	275	162
I_CA_01	14	350	256
I_OC_02	10	250	144
I_SM_01	12	300	187

se ha obtenido el tiempo invertido por el algoritmo en cada *frame*(t_{frame}) y la cantidad de *frames* que han sido evaluados (FE):

$$t = \frac{\sum (t_{frame})}{FE} \tag{3.1}$$

Precisión del algoritmo de *tracking* ($\mathbf{P}_{track}$). Mide como de bien sigue el algoritmo al objetivo. Para ello se han usado los datos etiquetados para comprobar la cantidad de píxeles que han sido identificados correctamente, o *true positives* (TP). Si la cantidad de píxeles supera un cierto *umbral* respecto a los píxeles total de la persona detectada (P_{total}), se considera que el *tracking* se ha realizado correctamente en ese *frame*. Realizamos esta evaluación para cada

frame para tener el numero total de *frames* correctamente procesados (FC):

$$FC = \sum \left(\frac{TP}{P_{total}} >= umbral \right), \tag{3.2}$$

donde el *umbral* utilizado en nuestros experimentos es de 0.8.

Una vez obtenido el número de *frames* en los que se ha hecho *tracking* correctamente (FC), se obtiene la precisión del algoritmo en dicha secuencia calculando la fracción de *frames* correctos respecto a los *frames* evaluados (FE):

$$P_{track} = \frac{FC}{FE}. \tag{3.3}$$

Precisión del detector ($\mathbf{P}_{det}$). Mide como de bien localiza personas el detector. Esta métrica permite saber si los algoritmos de *tracking* reciben una buena inicialización del detector, ya que si este no funciona bien para una secuencia dada, el algoritmo de *tracking* tampoco lo hará. Para su calculo se han contabilizado las detecciones correctas (TP) y las detecciones falsas (FP), en los *frames* en los que no se utilizase *tracking*, con los datos etiquetados y se ha obtenido la precisión del sistema (ratio entre las detecciones correctas de las detecciones totales):

$$P_{det} = \frac{TP}{TP + FP}. \tag{3.4}$$

En este caso, no se ha realizado el cálculo del *recall*, porque no interesa comprobar si el detector encuentra a todas las personas de una imagen (recall), sino que cuando detecta una persona, esta sea correcta (precisión) ya que es la inicialización que damos al *tracking*.

3.2.2. Experimentos realizados

A continuación se muestran y comparan los resultados obtenidos con todos los algoritmos descritos en la sección 3.1 salvo el algoritmo Goturn, ya que no se pudo ejecutar debido a un *bug* con este algoritmo en la versión 3.2 de OpenCV. También se decidió no utilizar el algoritmo de CamShift debido a que en pruebas preliminares se observó que requería muchos más recursos que los anteriores y a que era de mayor interés evaluar los otros algoritmos al haber sido incluidos en la versión 3.2 de OpenCV.

3.2.2.1. Evaluación tiempos de ejecución

Uno de los puntos más importantes a tener en cuenta a la hora de evaluar los algoritmos de *tracking* es el tiempo empleado en tratar cada *frame*. Dado que el sistema pretende realizar tareas de vigilancia en tiempo real, es necesario que el algoritmo elegido permitiese tratar varios *frames* por segundo para obtener un resultado aceptable.

Experimento 1: Tiempo por *frame* de todos los métodos aplicados en las secuencias de la evaluación inicial. Se ha calculado el tiempo medio que cada algoritmo emplea en los *frames* de cada secuencia de las detalladas en la Tabla 3.2. Se utilizan las métricas descritas en la sección 3.2.1.2, ecuación

Tabla 3.3: Tiempo por *frame* de los algoritmos de *tracking* en las distintas secuencias procesadas en la Raspberry Pi.

Secuencias	Boosting		MIL	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01	150	0,419	150	0,776
I_BS_02	100	0,363	100	0,734
I_CA_01	241	0,158	241	0,565
I_OC_02	100	0,369	100	0,631
I_SM_01	217	0,35	217	0,656
MEDIA		0,332		0,6724

Secuencias	KCF		MedianFlow	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01	150	0,149	150	0,038
I_BS_02	100	0,115	100	0,038
I_CA_01	225	0,931	225	0,038
I_OC_02	100	0,233	75	0,037
I_SM_01	217	0,379	142	0,038
MEDIA		0,361		0,038

Secuencias	TLD	
	Frames evaluados	Tiempo por *frame* (s)
I_BS_01	150	0,692
I_BS_02	100	0,823
I_CA_01	225	0,309
I_OC_02	120	0,825
I_SM_01	191	1,013
MEDIA		0,732

(3.1). Estos datos han sido medidos ejecutando en la Raspberry Pi y pueden ser observados en la Tabla 3.3. En esta tabla puede observarse en cada fila los resultados obtenidos para cada secuencia, los *frames* evaluados con cada algoritmo y el tiempo empleado en cada frame. Para más detalles, como los tiempos obtenidos en PC, véase el Anexo C.

Experimento 2: Tiempo por *frame* de los algoritmos elegidos. Una vez elegidos los algoritmos que mejor se comportan, se ha medido el tiempo por *frame* resultante de aplicar dichos algoritmos sobre todas las secuencias de interior disponibles, utilizando la ecuación (3.1). En la Tabla 3.4 se puede ver el tiempo medio de dichas secuencias obtenido con cada algoritmo en Raspberry Pi. Para más detalles, como el tiempo por *frame* para cada secuencia, o los resultados en PC, véase el Anexo C.

3.2.2.2. Evaluación de la precisión del detector

Para saber si la precisión de los algoritmos de *tracking* evaluados puede considerarse como una métrica valida, es necesario saber si han sido inicializados

Tabla 3.4: Tiempo medio por *frame* de los algoritmos KCF y MedianFlow en todas las secuencias en Raspberry Pi.

Tiempo medio por *frame*	MedianFlow	KCF
	0,038	0,255

Tabla 3.5: Precisión media del detector de personas sobre el subconjunto de secuencias de evaluación inicial.

Secuencias	Precisión
I_BS_01	1
I_BS_02	1
I_CA_01	1
I_OC_02	1
I_SM_01	0,67

correctamente. Para ello se ha calculado la precisión del detector, ecuación (3.4), en cada una de las secuencias descritas en la Tabla 3.2. En cada secuencia solo se ejecuta la detección en unos pocos *frames*: al arrancar, para inicializar el *tracking*, y cuando se detecta que el *tracking* ha terminado o ha perdido el objetivo, para iniciar el seguimiento de otro objetivo.

Los resultados obtenidos pueden observarse en la Tabla 3.5. Cada fila en esta Tabla corresponde con una secuencia procesada y la precisión del algoritmo de detección en los *frames* correspondientes. Se muestra el resultado común para todos los algoritmos de *tracking*. El *tracking* solo influye por el hecho de que puede perder el objetivo más o menos veces, y por tanto requerir lanzar el detector en más o menos *frames*. Sin embargo, en todos los casos, la precisión del detector es muy buena para todos los *frames* que necesitan ejecutar el detector, salvo en la secuencia *I_SM_01*, donde el detector conseguía una precisión un poco mas alta al utilizar los algoritmos MedianFlow y TLD.

3.2.2.3. Evaluación de la precisión del *tracking*

Para que el sistema pueda desempeñar las funciones de monitorización correctamente, es importante que el algoritmo elegido presente una buena precisión en las tareas de detección y seguimiento de las personas (en escenarios y casos de uso lo más realistas posible). Para ello se han utilizado las ecuaciones (3.2) y (3.3) para obtener la precisión de cada algoritmo en cada secuencia.

Experimento 1: Evaluación inicial. Este experimento usa las secuencias mencionadas en la Tabla 3.2, en la cual se detalla su duración, el número de *frames* de los que esta compuesta y los *frames* en los que aparecen personas a detectar. El objetivo es ver que métodos son capaces de hacer un correcto seguimiento de las personas que aparecen en estas secuencias. Cada secuencia ha sido ejecutada con cada uno de los algoritmos. No fue necesario realizar más ejecuciones, ya que estos algoritmos devuelven siempre el mismo resultado para una misma secuencia. En la Tabla 3.6 puede observarse la media de la precisión obtenida al aplicar los algoritmos sobre las secuencias elegidas. Para más detalles sobre la precisión de los algoritmos en cada secuencia véase el Anexo D.

Tabla 3.6: Precisión (P_{track}) media de los algoritmos de *tracking* en el subconjunto de secuencias de evaluación inicial (utilizando la ecuación (3.3).

Precisión media	Boosting	MIL
	0,55	0,49

Precisión media	KCF	MedianFlow
	0,67	0,41

Precisión media	TLD
	0,31

Tabla 3.7: Precisión (P_{track}) media de los algoritmos MedianFlow y KCF en todas las secuencias de interior, calculado con la ecuación (3.3).

Precisión media	KCF	MedianFlow
	0,383	0,327

Experimento 2: Evaluación completa de los algoritmos seleccionados. Los algoritmos seleccionados a partir de la evaluación inicial fueron evaluados en detalle siendo ejecutados en todas las secuencias de interior de LASIESTA. En la Tabla 3.7 puede observarse la media de las precisiones obtenidas al aplicar los algoritmos seleccionados sobre todas las secuencias y utilizar las ecuaciones (3.2) y (3.3). Solo fue necesario realizar esta medición una vez, al devolver siempre los mismos resultados sobre una misma secuencia. Para obtener más detalles sobre la precisión de estos algoritmos sobre cada secuencia de interior de LASIESTA, vease el Anexo D.

3.2.3. Discusión

Una vez obtenidos los resultados de las evaluaciones se han analizado dichos resultados para determinar cuál sería el algoritmo que mejor encaja con el sistema propuesto.

Por un lado, la evaluación en tiempo de los algoritmo ha demostrado que el algoritmo más rápido es MedianFlow (0,038 segundos de media) seguido por KCF (0,361 segundos de media). Ambos algoritmos han obtenido un buen rendimiento en PC, sin embargo, al ejecutarlos en la Raspberry Pi se observó que solo MedianFlow conservaba unos tiempos rápidos en comparación con los demás, siguiendo KCF siendo el segundo algoritmo más rápido a pesar de esto. Aunque pudiese parecer que el algoritmo de Boosting tiene un mejor tiempo medio que KCF en Raspberry Pi, esto se debe al mal rendimiento de KCF en la secuencia *I_CA*, en la que una persona se queda quieta durante unos segundos, indicando esto que a KCF le cuesta detectar que el elemento a seguir esta detenido.

Por otro lado la precisión del detector y de los algoritmos de *tracking*. Al observar los resultados del detector se pudo comprobar que tenía una precisión perfecta, detectando siempre a las personas que aparecen en cada secuencia, salvo en la secuencia *I_SM*, en la que se obtiene una precisión algo más baja al estar la cámara en movimiento. Por lo tanto los resultados obtenidos con los algoritmos de *tracking* son perfectamente validos, al haber recibido en casi todos los casos una inicialización correcta.

En lo que respecta a la precisión de los algoritmos de *tracking*, se comprobó

que el algoritmo que conseguía seguir al objetivo con éxito en más ocasiones era KCF (0,67) seguido por Boosting (0,55) y siendo MedianFlow (0,41) y TLD (0,31) los que obtenían resultados más pobres.

Finalmente al observar los resultados, se decidió que los algoritmos que eran más útiles en la plataforma eran KCF y MedianFlow. A pesar de que Boosting y MIL obtenían mejores resultados que MedianFlow, sus tiempos de ejecución en Raspberry Pi eran demasiado altos, lo cual hacía inviable usarlo en sistema de vigilancia que necesita procesar varios *frames* por segundo. Además, aunque MedianFlow tenía los segundos peores resultados, estos no eran tan bajos como para desestimar por completo el algoritmo, teniendo también en cuenta, que las secuencias en las que no había ningún elemento inesperado como cambios de luz repentinos, el algoritmo seguía obteniendo muy buenos resultados.

Debido a lo expuesto anteriormente, se decidió que se utilizaría el algoritmo MedianFlow en el prototipo por su gran tiempo de ejecución a pesar de que el algoritmo KCF obtenía unos resultados mucho mejores. Sin embargo, se decidió no descartar del todo el algoritmo KCF, ya que, aunque su tiempo de ejecución era bastante mayor que el de MedianFlow, este podía ser reducido obteniendo imágenes de menor resolución y empleando un menor número de *frames* por segundo.

Capítulo 4

Arquitectura del prototipo y experimentos de integración

Una vez evaluadas las componentes *hardware* y los algoritmos base, se ha desarrollado el prototipo de monitorización y vigilancia. Este capítulo describe el diseño de dicho prototipo, los experimentos de integración y las pruebas realizadas para comprobar la funcionalidad del mismo.

4.1. Diseño y construcción del prototipo

Los principales elementos que se utilizados para el diseño del prototipo han sido:

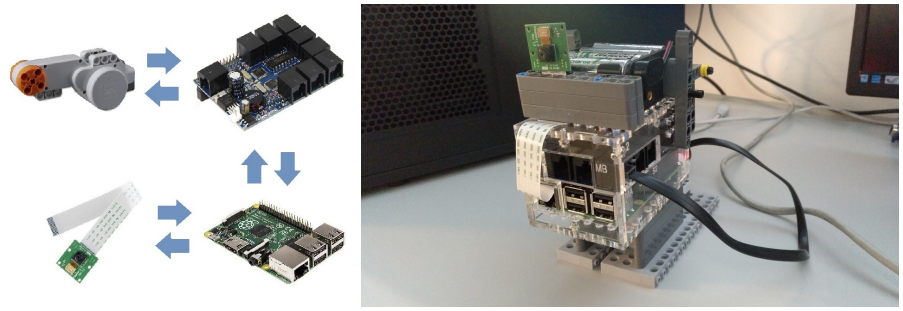
- Raspberry Pi.
- La placa Dexter Brickpi.
- Raspberry Pi Camera.
- Un motor LEGO Mindstorms.

Al conectar la Raspberry Pi y la placa Dexter Brickpi, junto a las piezas proporcionadas por esta ultima, se forma un módulo único que permite el uso de piezas de Lego.

Para el montaje final se utilizaron diversas piezas de lego para unir dichos elementos y crear una estructura que permitiera el correcto funcionamiento del prototipo. Estos elementos unidos permitieron construir un prototipo que pudiera realizar las tareas de visión además de tener cierta capacidad de movimiento gracias al motor, para poder orientar la cámara para seguir a los elementos detectados (Figura 4.1).

4.1.1. Evaluación de funcionalidad del prototipo

Una vez montado el prototipo, se realizaron varias pruebas para comprobar que el diseño elegido podía realizar de forma correcta las tareas de vigilancia.



(a) (b)

Figura 4.1: (a) Elementos principales del del prototipo construido. (b) Diseño final del prototipo.

Los principales puntos a probar fueron dos:

Visilibidad de la cámara. Al probar la cámara en la posición inicial, se comprobó que no se encontraba en una posición completamente correcta que afectaba a la visibilidad del dispositivo. Para solucionarlo se colocaron varias piezas de lego detrás de la cámara para que su sujeción fuera más robusta y no perdiera visibilidad si se inclinaba (Figura 4.2).

Potencia necesaria para el motor. En el momento de probar el motor, se descubrió que este soportaba bastante peso, por lo que fue necesario ajustar su potencia para conseguir que este pudiera realizar su movimiento sin problemas y que al mismo tiempo no fuese lo bastante rápido para poner en peligro la estructura.

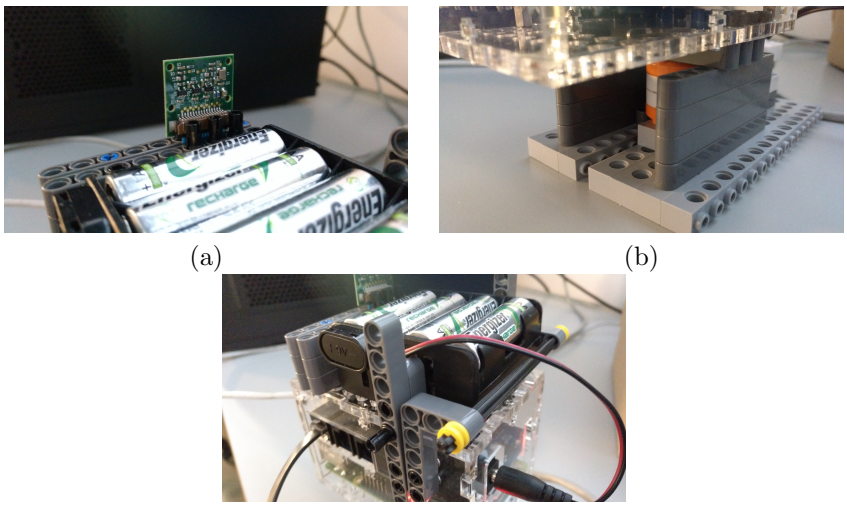
Además de lo anterior también se comprobó que el peso soportado por el motor hacia que al cabo del tiempo las piezas de lego se saliesen de su posición. Para solucionar este problema se colocaron otras piezas que ayudan al motor a soportar parte del peso de la estructura (Figura 4.2). Por ultimo, también se comprobó que era necesario construir una estructura auxiliar que sujetase la fuente de alimentación del prototipo, para que al moverse este, la fuente no se cayese de su posición (Figura 4.2).

Una vez solucionados estos problemas estructurales se consiguió que el prototipo pudiese tomar imágenes de su entorno y orientar la cámara sin ningún problema.

4.2. Arquitectura del sistema

El sistema diseñado, debe realizar tareas de vigilancia, siendo capaz de detectar determinados elementos en movimiento (personas) y monitorizarlos. Además el sistema enviará una imagen de los elementos detectados a Dropbox[1] para almacenar un registro de todos ellos con fácil acceso.

[1] `https://www.dropbox.com`



(a) (b)

(c)

Figura 4.2: (a) Estructura para mantener recta la cámara. (b) Piezas auxiliares para aguantar el peso de la Raspberry Pi y la placa. (c) Estructura para mantener la fuente de alimentación.

4.2.1. Módulos del sistema y flujo de ejecución

El sistema está implementado en python, y está organizado en los siguientes distintos módulos:

- **Módulo principal de control del sistema.** Inicia todo el sistema y obtiene los *frames* de la cámara del prototipo. Llama a las funciones de las otras clases para realizar detección o *tracking* además de enviar *frames* a Dropbox.
- **Módulo de Detección.** Se encarga de detectar a personas en los *frames* obtenidos por la clase principal, para determinar los objetivos a monitorizar.
- **Módulo de Seguimiento.** Realiza el *tracking* sobre los *frames* y el objetivo que recibe de la clase principal.
- **Módulo de control de motor.** Mueve el motor del prototipo en función de donde se encuentre el objetivo a seguir para re-orientar la cámara y mantenerlo lo más centrado posible en la imagen.

En la Figura 4.3 se puede ver un diagrama de las clases principales de este sistema. Los distintos módulos interactúan y se ejecutan siguiendo el diagrama resumen de la Figura 4.4. En concreto, las fases principales son:

- **Selección del objeto a seguir.** En primer lugar, el sistema se inicia, y pasa a aplicar detección sobre todos los *frames* que obtiene. Una vez que detecta algo se sube el *frame* en el que se ha detectado a Dropbox, de forma concurrente, para evitar ralentizar al programa principal.

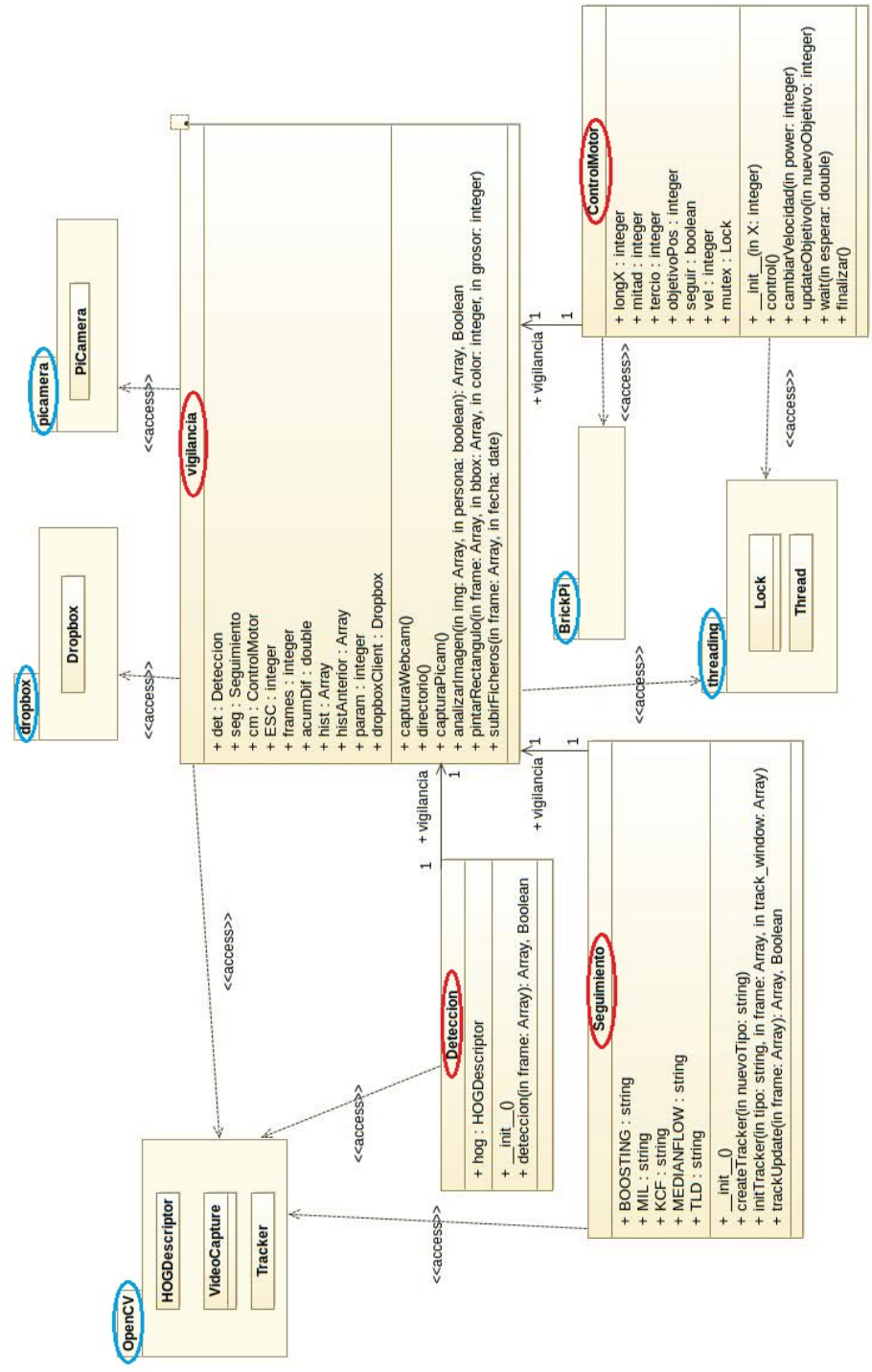


Figura 4.3: Diagrama de clases del sistema. En rojo los módulos implementados para el sistema, en azul los módulos y paquetes ya implementados utilizados en el sistema.

- **Seguimiento del objeto.** Al detectar algo, se pasa a aplicar el algoritmo de *tracking* sobre los siguientes *frames* para realizar una monitorización de dicho elemento. De los resultados del algoritmo de *tracking*, se obtienen sus distintos histogramas, para realizar una comparación entre ellos y así saber si el elemento a seguir se encuentra en movimiento, esta completamente quieto o se ha perdido. Si el elemento se hubiera perdido o se encontrase completamente quieto, se volvería a aplicar detección sobre los siguientes *frames*.

- **Movimiento del motor** Concurrentemente, se mueve el motor en función de la posición en la que se encuentre el elemento monitorizado. Si este se encuentra muy a la izquierda o a la derecha del *frame* actual, se moverá el motor para orientar la cámara hacia el mismo. Para realizar esta tarea, se divide el tamaño de los *frames* en el eje x en 3 partes, si el objetivo se encuentra en la zona derecha o izquierda, se mueve el motor. La posición del objetivo se actualiza cada vez que se aplica el algoritmo de *tracking*. Si no hay nada a seguir, o el elemento se encuentra en una posición central, el motor no se mueve.

4.2.2. Ejecución del sistema

Para ejecutar el sistema es necesario tener los ficheros *vigilancia.py*, *deteccion.py*, *seguimiento.py* y *controlMotor.py* en la misma ubicación. Una vez que se tienen todos los ficheros, el programa puede ser ejecutado simplemente con el comando *python vigilancia.py*, ejecutándolo así con la configuración por defecto. Se pueden encontrar más detalles sobre las distintas configuraciones del sistema en el Anexo E.

Durante la ejecución del sistema se mostrará el vídeo analizado en pantalla, ya sea capturado en tiempo real, o leído desde directorio. En las imágenes mostradas se podrá apreciar la detección y el *tracking* realizado sobre dichas imágenes, pudiendo distinguir claramente cuando se esta ejecutando cada algoritmo:

- En el momento en que el algoritmo de detección encuentre algo, ese área de la imagen sera señalada mediante un recuadro rojo (Figura 4.5(a)).

- Al aplicar el algoritmo de *tracking* se señalara el área sobre la que el algoritmo realiza el seguimiento mediante un recuadro azul (Figura 4.5(b)).

4.3. Experimentos de Integración

Finalmente, se realizaron las pruebas de integración con el prototipo para realizar tareas de vigilancia y monitorización. Estos experimentos se dividieron principalmente en dos partes: por un lado experimentos en tiempo real, con el prototipo realizando tareas de vigilancia en un entorno determinado; por otro lado pruebas con vídeos grabados previamente desde el prototipo en entornos realistas.

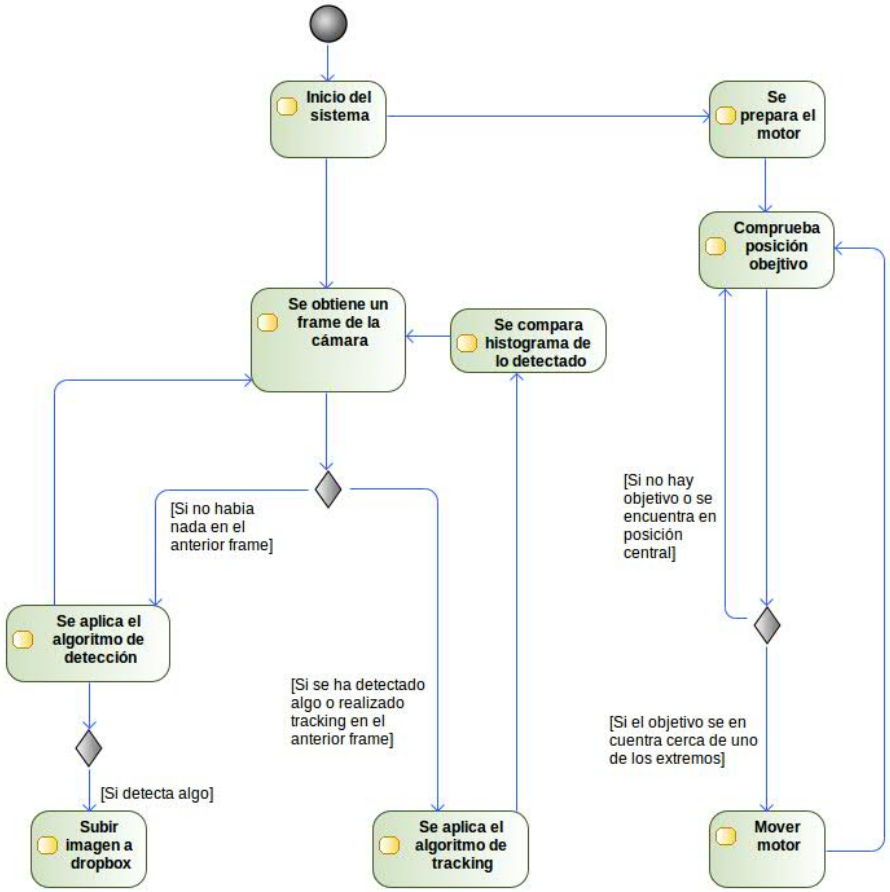


Figura 4.4: Diagrama del funcionamiento del sistema.

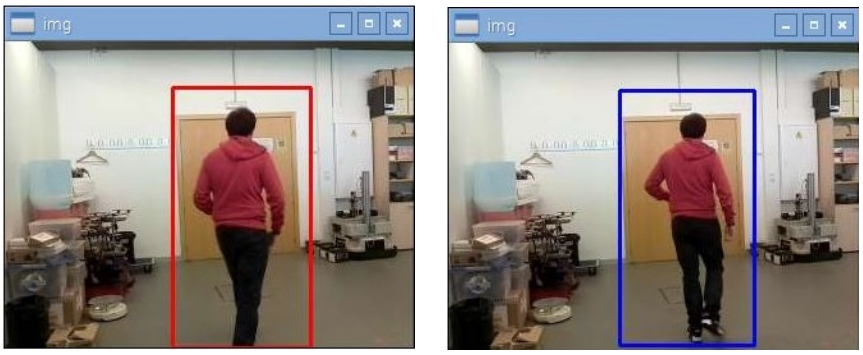
4.3.1. Experimentos en tiempo real

Estos experimentos en diversos lugares con distintas personas. A continuación se detallan los resultados obtenidos en las diferentes fases del sistema.

Evaluación del detector. La ejecución del detector no es muy rápida (unos 0.3 segundos por *frame* procesado). Sin embargo, esto no afecta mucho al sistema, porque es suficiente para detectar a las personas que entran en el ángulo de visión de forma satisfactoria e inicializar el *tracking*.

El objetivo es tener una manera automática de inicializar el seguimiento, así que el detector utilizado no ha sido el centro de estudio. El que se utiliza es uno sencillo disponible en OpenCV y como se comenta, que se pueda ejecutar con suficiente rapidez, aunque solo funciona de manera robusta con buenas condiciones (vista de cuerpo entero, sin muchas oclusiones de la persona).

Evaluación del *tracking*. El algoritmo de *tracking*, necesita el buen funcionamiento del detector para inicializarse (es decir, obtener un objetivo a seguir).



(a) (b)

Figura 4.5: El sistema señala el elemento localizado por el algoritmo de detección con un recuadro rojo (a) o con un recuadro azul si ya ha entrado en la fase de *tracking* (b).

La ejecución del algoritmo de *tracking* es más rápida que el detector, permitiendo procesar mayor numero de *frames* por segundo durante el modo de *tracking*. Sin embargo, los resultados obtenidos no fueron muy buenos, llegando solamente a realizar un seguimiento aceptable del objetivo en aproximadamente un 20 % de los casos. El problema suele ser que el sistema pierde al objetivo antes de que este saliese del ángulo de visión de la cámara. También se han realizado pruebas similares con el algoritmo KCF, al haber obtenido mejores resultados en las pruebas anteriores (sección 3.2.2). Sin embargo, los resultados no mejoraron demasiado, siendo muy similares a los de MedianFlow.

Para intentar mejorar los resultados, se redujo la resolución de los *frames* obtenidos. Sin embargo, aunque se observo una mejora del tiempo empleado en *tracking*, no fue suficiente como para mejorar los resultados del *tracking*, y sin embargo empeoraban los resultados del detector inicial.

La conclusión de estas múltiples pruebas realizadas en directo son que la versión actual del prototipo realiza seguimientos cortos por problemas de capacidad de cómputo. Esta hipótesis se verifica en la sección siguiente, con varios experimentos sobre secuencias grabadas evaluando los resultados con mayor número de *frames* procesados por segundo.

Control del motor. Funciona de forma bastante precisa, siendo capaz de orientar la cámara hacia el objetivo en aproximadamente el 100 % de los casos, siempre que se realizaba *tracking* correctamente. El único fallo que se le encontró, fue que al moverse producía un movimiento muy brusco en la cámara lo cual producía que el algoritmo de seguimiento perdiese al objetivo.

4.3.2. Experimentos con vídeos propios

El objetivo de este experimento es comprobar el funcionamiento del sistema sin las limitaciones de realizarlo en tiempo real. Para ello se grabaron varios vídeos en entornos realistas desde el prototipo, y así poder evaluar la calidad de los resultados cuando podemos procesar más *frames* por segundo. Estos vídeos se grabaron a 32 *frames* por segundo y con una resolución de 320x240, en dos

Escenario 1	
Secuencias	**Descripción**
Secuencia 1	El usuario 1 se mueve de un lado a otro del escenario.
Secuencia 2	El usuario 1 se dirige al fondo del escenario y sale del mismo para luego volver a entrar.
Secuencia 3	El usuario 2 aparece desde el primer *frame* en la escena y se mueve por el escenario con un movimiento de zigzag.
Secuencia 4	El usuario 3 se dirige al fondo del escenario y sale del mismo.
Secuencia 5	Los usuario 2 y 3 aparecen juntos en la escena y dan vueltas por el escenario.
Secuencia 6	El usuario 3 pasa rápidamente de un lado a otro de la escena.
Escenario 2	
Secuencias	**Descripción**
Secuencia 7	El usuario 1 viene del fondo de la escena.
Secuencia 8	El usuario 4 se dirige al fondo de la escena.
Secuencia 9	El usuario 4 da vueltas por el escenario.
Secuencia 10	El usuario 5 viene del fondo de la escena y luego se retira.
Secuencia 11	El usuario 1 recorre el escenario con condiciones de luz diferentes a las de otras secuencias.

Tabla 4.1: Descripción de las secuencias grabadas con la cámara del prototipo.

escenarios diferentes con distintas personas. Estas secuencias están descritas en la Tabla 4.1. Las Figuras 4.6 y 4.7 muestran imágenes de ejemplo de las secuencias en los escenarios 1 y 2, respectivamente.

Una muestra de los resultados obtenidos de ejecutar el detector y el algoritmo MedianFlow sobre las secuencias grabadas en los escenarios 1 y 2 pueden verse en las Tablas 4.2 y 4.3 respectivamente. En las filas de la figura pueden verse para cada una de las secuencias elegidas, la persona encontrada por el detector y el elemento sobre el que se esta realizando el seguimiento, al comienzo de empezar a aplicar el algoritmo de *tracking* y justo antes de que el objetivo abandone la escena. Las Tablas 4.4 y 4.5 muestran resultados similares al aplicar el *tracking* KCF en los escenarios 1 y 2 respectivamente.

Evaluación del detector. Se observaron, al igual que en los experimentos en tiempo real buenos resultados. Siendo solo necesario que la persona a detectar sea vista de cuerpo entero sin que nada obstaculice el ángulo de visión hacia dicha persona y unas condiciones aceptables de luz. Se puede observar en las Tablas 4.2 y 4.3 como el detector solamente ha fallado en las secuencias 6 y 11 en las que no ha detectado nada, debido a que el objetivo a detectar estaba borroso y a las distintas condiciones de luz, respectivamente.

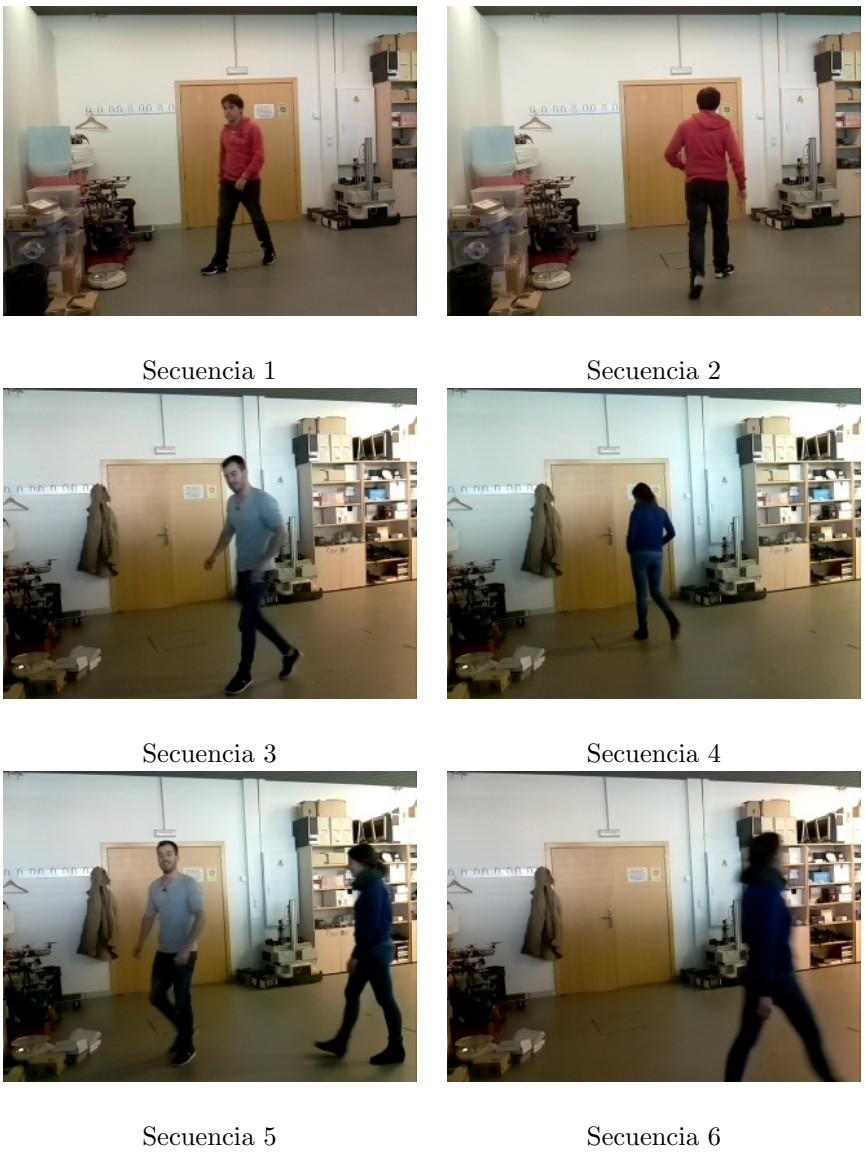


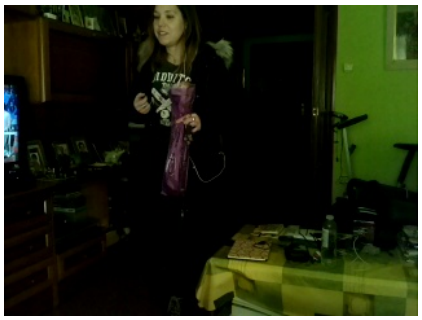
Figura 4.6: Imágenes tomadas con la Raspberry Pi Camera del sistema en el **escenario 1**.

Evaluación del *tracking*. En general, se observaron mejores resultados que en los experimentos en tiempo real de la Sección 4.3.1, debido a que el sistema pudo procesar todos los *frames* de las secuencias, sin las limitaciones de tener que procesar los *frames* en tiempo real. Para comprobar el funcionamiento de los algoritmos elegidos, y a falta de datos etiquetados de estas secuencias, simplemente se observó en cada secuencia si el algoritmo era capaz de seguir al objetivo, y durante cuanto tiempo podía seguirlo. Estos experimentos no incluyeron finalmente las secuencias 6 y 11 porque el detector no encontró ninguna persona en dichas secuencias para inicializar el *tracking*. Por un lado, el

Secuencia 7

Secuencia 8

Secuencia 9



Secuencia 10

Secuencia 11

Figura 4.7: Imágenes tomadas con la Raspberry Pi Camera del sistema en el **escenario 2**.

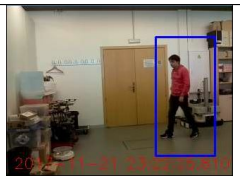
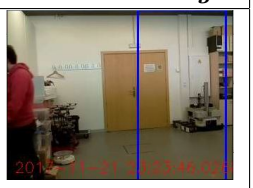
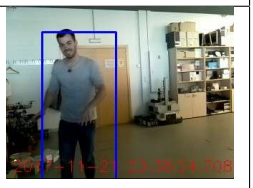
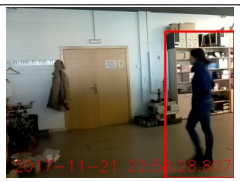
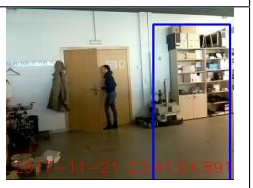
Escenario 1 (MedianFlow)			
Secuencias	**Detector**	**Inicio *tracking***	**Final *tracking***
Secuencia 1			
Secuencia 2			
Secuencia 3			
Secuencia 4			
Secuencia 5			

Tabla 4.2: Ejecución del sistema (MedianFlow) en secuencias del **escenario 1**.

algoritmo de MedianFlow solo funcionó correctamente en el 50 % de los casos aproximadamente, perdiendo en el resto de los casos al objetivo poco después de inicializarse. Por otro lado el algoritmo KCF, consiguió muy buenos resultados en todas las secuencias, salvo en la 8, en la que fue incapaz de seguir al objetivo. En todas la demás logró seguir al objetivo sin perderlo. Sin embargo, el tiempo que requirió la ejecución de KCF fue aproximadamente 3 veces mayor que el empleado por MedianFlow.

Los experimentos anteriores demuestran que el tiempo empleado por los algoritmos de *tracking* para cada *frame* es un cuello de botella, ya que son muy altos para que la Raspberry Pi pueda procesar en tiempo real todos los *frames* capturados por la cámara. Por lo tanto, el sistema actual ahora mismo funciona en vivo correctamente cuando el objetivo a seguir no ha variado demasiado su posición entre dos *frames* consecutivos procesados por el sistema.

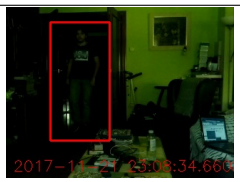
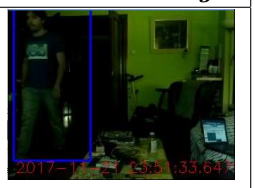
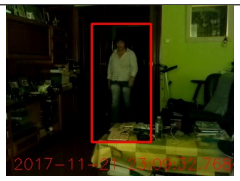
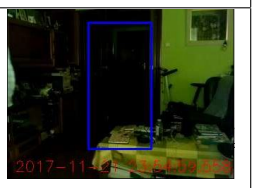
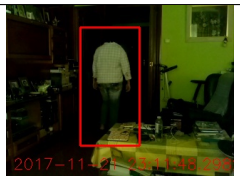
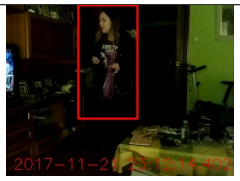
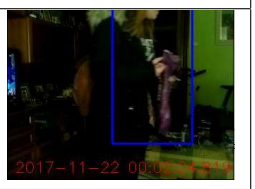
Escenario 2 (MedianFlow)			
Secuencias	**Detector**	**Inicio *tracking***	**Final *tracking***
Secuencia 7			
Secuencia 8			
Secuencia 9			
Secuencia 10			

Tabla 4.3: Ejecución del sistema (MedianFlow) en secuencias del **escenario 2**.

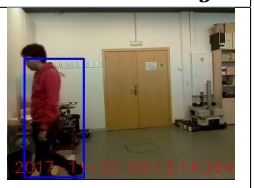
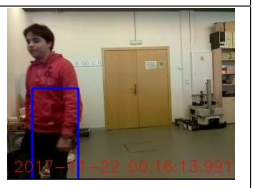
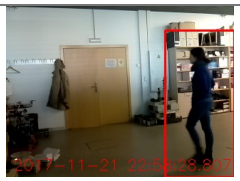
Escenario 1 (KCF)			
Secuencias	**Detector**	**Inicio *tracking***	**Final *tracking***
Secuencia 1			
Secuencia 2			
Secuencia 3			
Secuencia 4			
Secuencia 5			

Tabla 4.4: Ejecución del sistema (KCF) en secuencias del **escenario 1**.

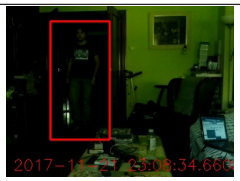
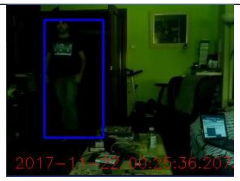
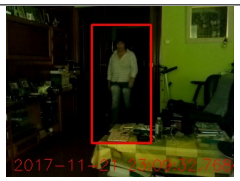
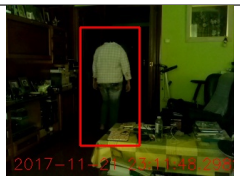
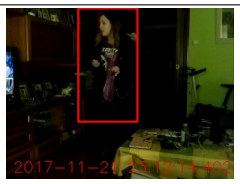
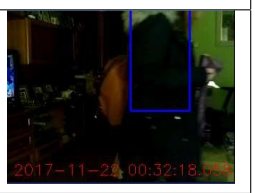
Escenario 2 (KCF)			
Secuencias	**Detector**	**Inicio *tracking***	**Final *tracking***
Secuencia 7			
Secuencia 8			
Secuencia 9			
Secuencia 10			

Tabla 4.5: Ejecución del sistema (KCF) en secuencias del **escenario 2**.

Capítulo 5

Conclusiones

5.1. Conclusión

Los dos objetivos principales de este proyecto se han alcanzado satisfactoriamente.

Por un lado, se han evaluado distintas plataformas de bajo coste para su uso en los campos de robótica y visión computador, así como su uso en conjunto con sensores y motores de LEGO Mindstorms. Estas evaluaciones servirán para marcar el desarrollo de futuros proyectos y prácticas de visión por computador sobre plataformas de bajo coste en el grupo de robótica, percepción y tiempo real de la Universidad de Zaragoza. Al haber realizado diversas evaluaciones de plataformas proporcionadas por dicho grupo para su uso en este campo de la visión por computador.

Por otro lado, se ha diseñado y evaluado un prototipo de aplicación utilizando dichas plataformas para una tarea realista.

Se han evaluado distintos algoritmos para su uso en sistemas de vídeo vigilancia y monitorización con las plataformas seleccionadas. De estos algoritmos se han seleccionado aquellos que han mostrado un mejor funcionamiento con las plataformas de bajo coste elegidas.

Finalmente, se ha diseñado e implementado un prototipo de sistema de vídeo vigilancia con las plataformas de bajo coste y los algoritmos elegidos. Con este prototipo se han realizado diversos experimentos para comprobar su funcionamiento. Si bien los resultados obtenidos no han sido perfectos, se ha demostrado que este tipo de plataformas son capaces de ejecutar aplicaciones visión por computador obteniendo resultados muy aceptables y prometedores para su uso en trabajos futuros.

5.2. Problemas encontrados

A continuación se detallan los principales problemas que se han encontrado durante la realización del proyecto y como se han resuelto o concluido:

- Las pruebas realizadas con la placa PiStorms sufrieron varios retrasos

debido a que la conexión con su fuente de alimentación era susceptible a roturas. Esta fue una de las razones que motivó la elección de Dexter BrickPi para el sistema, en lugar de PiStorms.

- Las pruebas con la cámara NXTCAMv4 también fueron otro punto que generó ciertos retrasos, ya que al no ser un sensor soportado oficialmente por los fabricantes de las placas, había poca documentación y finalmente no se consiguió que funcionaran correctamente. Por lo tanto esta cámara fue descartada como se ha comentado anteriormente.
- La mayoría de los algoritmos de *tracking* evaluados no detectan cuando han perdido a su objetivo y deben finalizar su ejecución. Por este motivo, para poder realizar una evaluación correcta y posteriormente un sistema de vigilancia funcional, fue necesario implementar un método que detecte esta situación, para finalizar el seguimiento una vez perdido el objetivo.

5.3. Trabajo futuro

Como trabajo futuro se han propuesto algunas ideas para mejorar el presente proyecto:

- Proveer de una fuente de alimentación más duradera. El sistema actual precisa del uso de una fuente alimentación para poder mover el motor, por lo que precisaría de una que le diese más autonomía que la actualmente usada, la cual consiste en el uso de 8 pilas AA.
- Implementación de un método de detección más sofisticado. El método empleado funciona bien en los entornos evaluados, pero hoy en día hay métodos de detección mucho más sofisticados que permitirían inicializar el algoritmo en entornos más variados, gracias a una detección de personas más robusta.
- Optimización de los algoritmos de *tracking* seleccionados. Como ya se ha mencionado en los anteriores capítulos, el tiempo utilizado en aplicar los algoritmos de *tracking* afecta a la cantidad de *frames* por segundo que se pueden capturar y procesar, y empeora el funcionamiento en tiempo real del sistema, por lo que una mejora en los mismos, mejoraría sustancialmente el sistema.
- Evaluación del sistema con otras plataformas. Otra opción para mejorar el sistema, sería su evaluación con otras plataformas *hardware*, como por ejemplo una versión más actual de Raspberry Pi, que tengan una mayor capacidad de cómputo.
- Mejorar el sistema de movimiento del motor para que no realice movimientos bruscos y pueda proporcionar una mayor amplitud de movimiento a la cámara.

Bibliografía

[1] Wilson Feipeng Abaya, Jimmy Basa, Michael Sy, Alexander C Abad, and Elmer P Dadios. Low cost smart security camera with night vision capability using raspberry pi and opencv. In *Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), 2014 International Conference on*, pages 1–6. IEEE, 2014.

[2] G Senthilkumar, K Gopalakrishnan, and V Sathish Kumar. Embedded image capturing system using raspberry pi system. *International Journal of Emerging Trends & Technology in Computer Science*, 3(2):213–215, 2014.

[3] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[4] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[5] Helmut Grabner, Michael Grabner, and Horst Bischof. Real-time tracking via on-line boosting. In *Bmvc*, volume 1, page 6, 2006.

[6] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 983–990. IEEE, 2009.

[7] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *European conference on computer vision*, pages 702–715. Springer, 2012.

[8] Martin Danelljan, Fahad Shahbaz Khan, Michael Felsberg, and Joost Van de Weijer. Adaptive color attributes for real-time visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1090–1097, 2014.

[9] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *Pattern recognition (ICPR), 2010 20th international conference on*, pages 2756–2759. IEEE, 2010.

[10] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409–1422, 2012.

[11] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *European Conference on Computer Vision*, pages 749–765. Springer, 2016.

[12] Gary R Bradski. Real time face and object tracking as a component of a perceptual user interface. In *Applications of Computer Vision, 1998. WACV'98. Proceedings., Fourth IEEE Workshop on*, pages 214–219. IEEE, 1998.

[13] Carlos Cuevas, Eva María Yáñez, and Narciso García. Labeled dataset for integral evaluation of moving object detection algorithms: Lasiesta. *Computer Vision and Image Understanding*, 152:103–117, 2016.

Apéndice A

Dexter BrickPi

Como ya se ha comentado anteriormente Dexter BrickPi es una placa de adaptación que permite el uso de sensores y motores a una Raspberry Pi, permitiendo así la construcción con estas plataformas de sistemas roboticos y de visión por computador entre otros. En la Figura A.1 puede verse tanto la placa Dexter BrickPi, como la placa montada con la Raspberry Pi.

A.1. Alimentación de la placa

Una vez que Dexter BrickPi esta conectada a la Raspberry Pi existen dos formas de alimentar al sistema que forman:

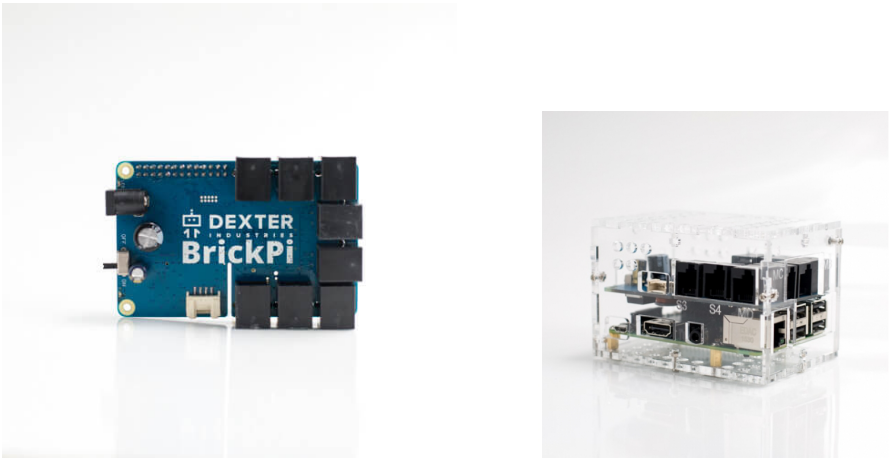
Alimentación a través de la Raspberry Pi. Se puede alimentar el sistema que forman por el puerto micro-USB (Figura A.2) de la Raspberry Pi mediante un cargador. Este método de alimentación sin embargo, no permitirá el uso de motores ya que la Raspberry Pi no recibe suficiente energía por el puerto micro-USB para alimentarlos.

Alimentación a través de Dexter BrickPi. El sistema puede ser alimentado mediante Dexter BrickPi (Figura A.3). Este método es el recomendado si se va a hacer uso de motores, ya que es el único medio por el que el sistema puede recibir energía suficiente para hacerlo funcionar.

A.2. Uso de Dexter BrickPi

Para utilizarlo es necesario tener instalado el Sistema Operativo para Dexter BrickPi, disponible en su página web. Hecho esto es posible utilizar Dexter BrickPi de dos formas:

Uso de monitor y otros periféricos. Conectando un monitor, un ratón y un teclado a la Raspberry Pi, se puede acceder y controlar el sistema de forma normal.



(a) (b)

Figura A.1: (a) Placa Dexter BrickPi. (b) Dexter BrickPi montada con Raspberry Pi.

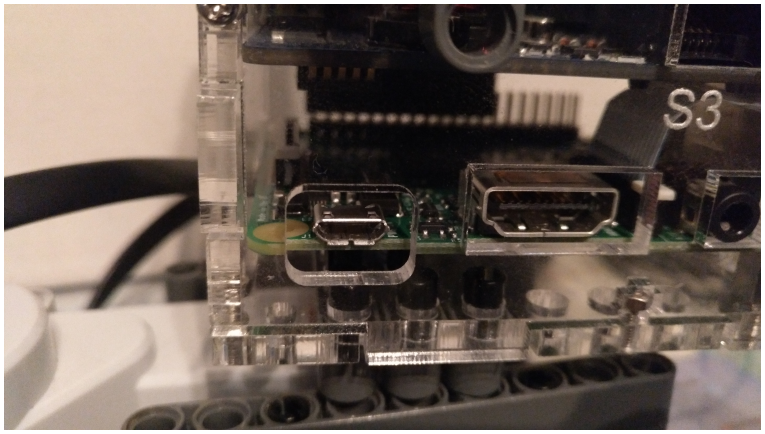


Figura A.2: Puerto micro-USB de la Raspberry Pi.

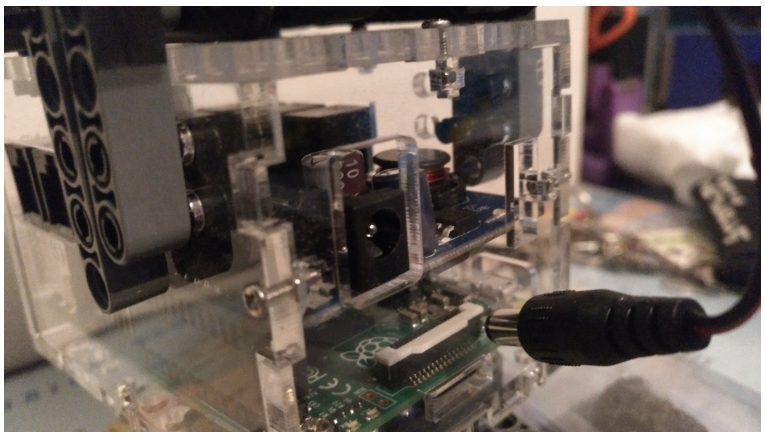


Figura A.3: Enchufe de Dexter BrickPi.

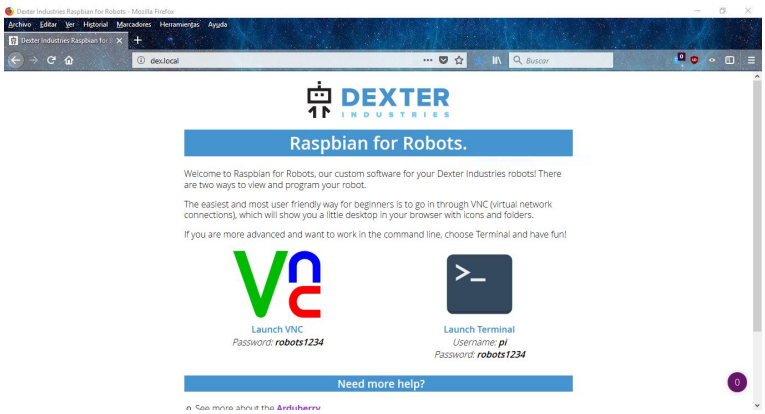


Figura A.4: Acceso mediante internet.

Mediante internet. Usando una antena *wifi* o un cable Ethernet. El Sistema Operativo que se usa con Dexter BrickPi tiene disponible un servidor VNC que permitirá la conexión remota con el sistema. Para acceder al mismo solamente será necesario escribir en un navegador la dirección `dex.local`, elegir como utilizar el sistema, mediante escritorio o por linea de comandos (Figura A.4) e introducir la contraseña para acceder.

Una vez que se esta dentro el sistema se maneja como cualquier otra Raspberry Pi u ordenador ya sea mediante escritorio como se muestra en la Figura A.5, como por línea de comandos como se ve en la Figura A.6.

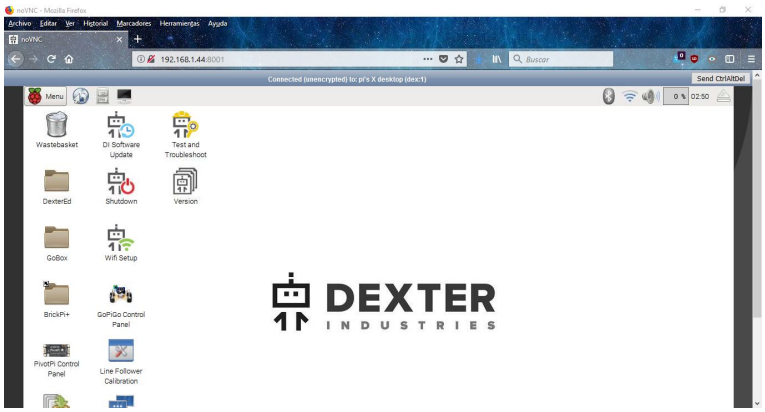


Figura A.5: Escritorio del sistema.

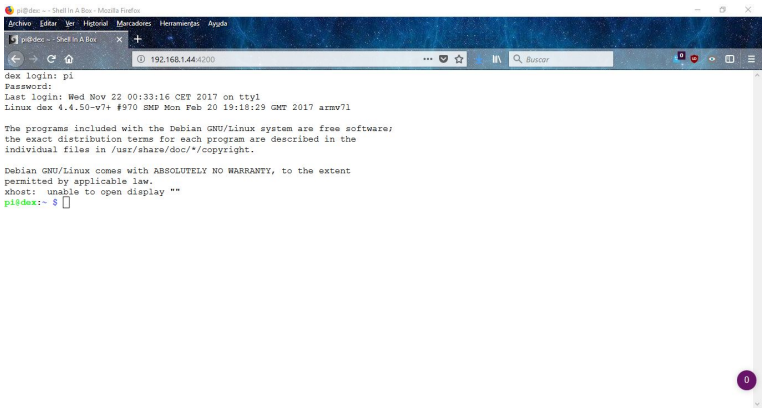


Figura A.6: Control del sistema por línea de comandos.

Apéndice B

Conjunto de datos LASIESTA

Las secuencias de LASIESTA [13] han sido usadas para realizar las evaluaciones de distintos algoritmos. Este conjunto de datos esta formado tanto por secuencias de interior como de exterior,sin embargo, debido al enfoque del proyecto, solo se han utilizado las secuencias de interior para evaluación.

Las secuencias tienen una resolución de 352x288 y están etiquetadas mediante segmentación que indica que píxeles de la imagen corresponden a una persona. A su vez hay secuencias de varios tipos que son descritas en la Tabla B.1. Las principales características de las secuencias están descritas en la Tabla B.2, en la cual en cada fila se indica, para cada secuencia, su duración en segundos, la cantidad de *frames* por los que está formada y en cuantos *frames* aparece una persona.

A continuación de la figuras B.1 a la B.8, se muestran varios ejemplos de los distintos tipos de secuencias, así como de sus datos etiquetados.

Tabla B.1: Tipos de secuencias de LASIESTA

Tipo de Secuencia	**Descripción**
SI	Secuencias simples, sin nada especial.
OC	Secuencias con oclusión de objetos en movimiento.
IL	Secuencias con cambios de iluminación.
MB	Secuencias en las que se abandonan objetos o estos son sustraidos del lugar.
BS	Secuencias con objetos en movimiento desde el primer frame.
MC b	Secuencias grabadas con cámara no estática.
SM	Secuencias que simulan distintos movimientos de la cámara.

Tabla B.2: Datos de todas las secuencias de LASIESTA

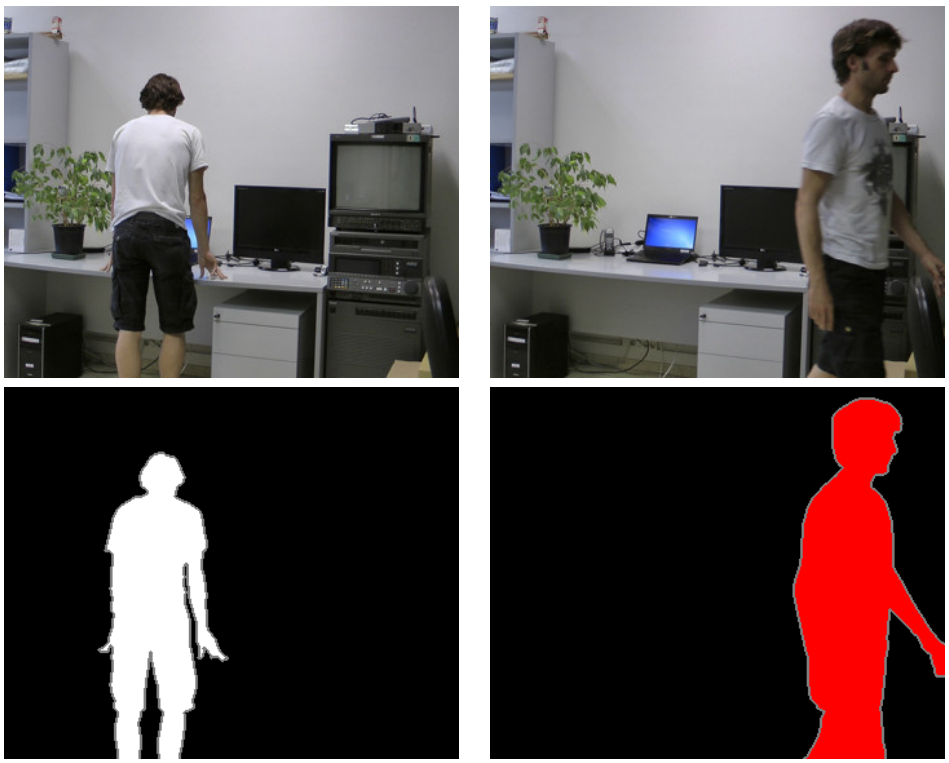
Secuencias	Duración (segundos)	*Frames*	Frames con personas
I_BS_01	11	275	183
I_BS_02	11	275	162
I_CA_01	14	350	256
I_CA_02	21	525	326
I_IL_01	12	300	102
I_IL_02	21	525	327
I_MB_01	18	450	356
I_MB_02	14	350	224
I_MC_01	12	300	128
I_MC_02	10	250	104
I_OC_01	10	250	141
I_OC_02	10	250	144
I_SI_01	12	300	209
I_SI_02	12	300	207
I_SM_01	12	300	187
I_SM_02	12	300	233
I_SM_03	12	300	236
I_SM_04	12	300	175
I_SM_05	12	300	175
I_SM_06	12	300	175
I_SM_07	12	300	176
I_SM_08	12	300	176
I_SM_09	12	300	175
I_SM_10	12	300	176
I_SM_11	12	300	176
I_SM_12	12	300	178

I_BS_01

I_BS_02

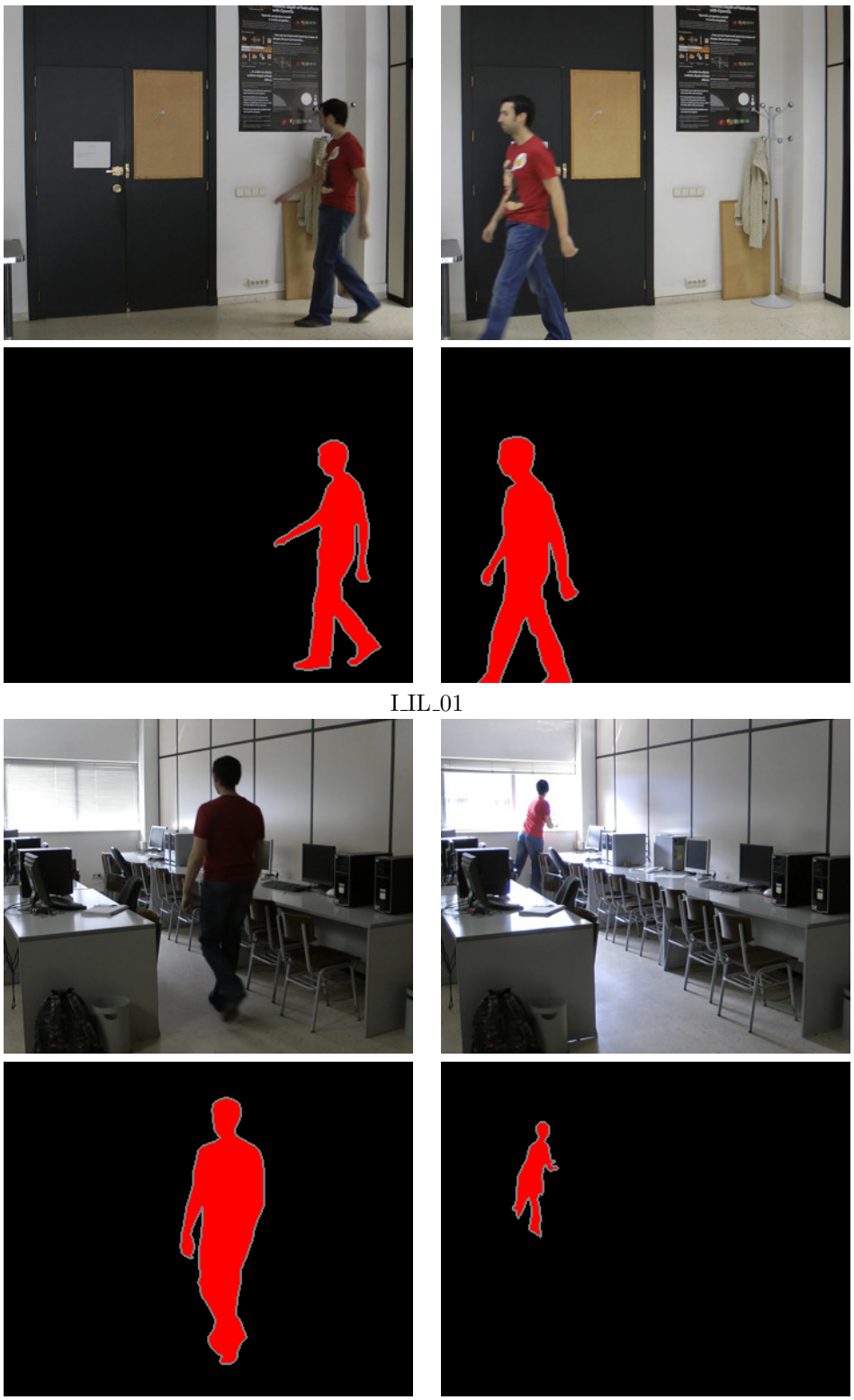
Figura B.1: Ejemplos de las secuencias I_BS_01 y I_BS_02.

I_CA_01



I_CA_02

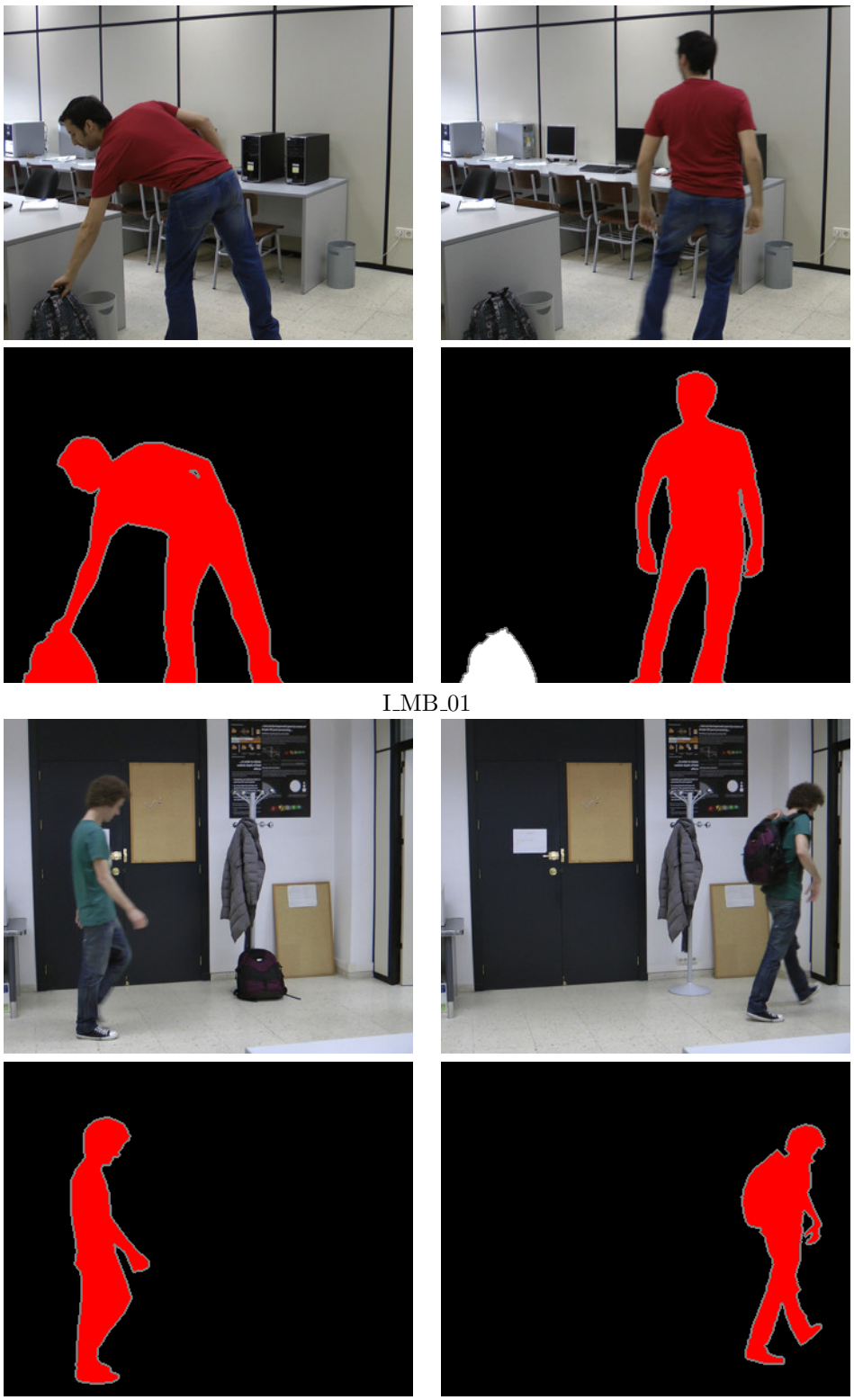
Figura B.2: Ejemplos de las secuencias I_CA_01 y I_CA_02.



I_IL_01

I_IL_02

Figura B.3: Ejemplos de las secuencias I_IL_01 y I_IL_02.



I_MB_01

I_MB_02

Figura B.4: Ejemplos de las secuencias I_MB_01 y I_MB_02.

I_MC_01

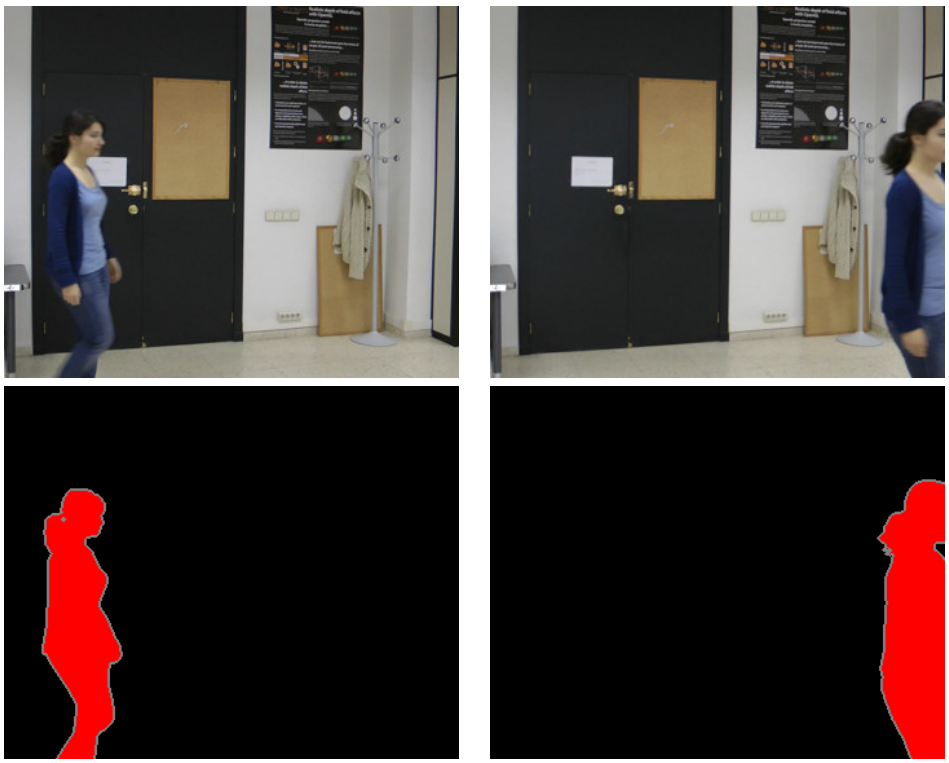
I_MC_02

Figura B.5: Ejemplos de las secuencias I_MC_01 y I_MC_02.

I_OC_01

I_OC_02

Figura B.6: Ejemplos de las secuencias I_OC_01 y I_OC_02.



I_SI_01

I_SI_02

Figura B.7: Ejemplos de las secuencias I_SI_01 y I_SI_02.

I_SM_01

I_SM_02

Figura B.8: Ejemplos de las secuencias I_SM_01 y I_SM_02.

Apéndice C

Evaluación de los tiempos de *tracking*

En este Anexo se muestran todos los datos obtenidos al medir el tiempo de ejecución de los algoritmos de tracking mediante la ecuación 3.1. Los tiempos han sido obtenidos tanto en Raspberry Pi como en PC y con dos resoluciones distintas, la original (352x288) y con resolución reducida al 60 %. Los detalles de las distintas secuencias pueden encontrar se en la Tabla B.2.

A continuación se detalla el contenido de las distintas tablas:

- La Tabla C.1 y C.2 muestran los tiempos obtenidos en PC al utilizar todos los algoritmos de *tracking* a evaluar sobre un grupo de secuencias seleccionadas con la resolución original y con resolución al 60 % respectivamente.
- La Tabla C.3 y C.4 muestran los tiempos obtenidos en Raspberry Pi al utilizar todos los algoritmos de *tracking* a evaluar sobre un grupo de secuencias seleccionadas con la resolución original y con resolución al 60 % respectivamente.
- Las Tablas C.5 y C.6 muestran los tiempos obtenidos, con resolución original y al 60 % respectivamente, de aplicar KCF y MedianFlow sobre cada una de las secuencias de interior en Raspberry Pi.
- Las Tablas C.7 y C.8 muestran los tiempos obtenidos, con resolución original y al 60 % respectivamente, de aplicar KCF y MedianFlow sobre cada una de las secuencias de interior en PC.

Tabla C.1: Tiempo por *frame* de los algoritmos de *tracking* (PC).

Secuencias	Boosting		MIL	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01	150	0,046	150	0,149
I_BS_02	100	0,041	100	0,132
I_CA_01	241	0,018	241	0,083
I_OC_02	100	0,043	100	0,117
I_SM_01	217	0,041	192	0,136
MEDIA		0,038		0,123

Secuencias	KCF		MedianFlow	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01	150	0,009	150	0,004
I_BS_02	100	0,007	100	0,005
I_CA_01	225	0,046	225	0,005
I_OC_02	100	0,015	75	0,004
I_SM_01	217	0,018	142	0,005
MEDIA		0,019		0,005

Secuencias	TLD	
	Frames evaluados	Tiempo por *frame* (s)
I_BS_01	150	0,288
I_BS_02	100	0,267
I_CA_01	225	0,176
I_OC_02	100	0,313
I_SM_01	141	0,305
MEDIA		0,269

Tabla C.2: Tiempo por *frame* de los algoritmos de *tracking* con resolución al 60 % (PC).

Secuencias	Boosting		MIL	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01-60 %	200	0,027	200	0,099
I_BS_02-60 %	0	0	0	0
I_CA_01-60 %	100	0,03	100	0,134
I_OC_02-60 %	100	0,028	100	0,12
I_SM_01-60 %	136	0,029	136	0,125
MEDIA		0,023		0,096

Secuencias	KCF		MedianFlow	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01-60 %	175	0,007	200	0,003
I_BS_02-60 %	0	0	0	0
I_CA_01-60 %	75	0,007	100	0,003
I_OC_02-60 %	100	0,008	75	0,003
I_SM_01-60 %	161	0,008	135	0,004
MEDIA		0,006		0,003

Secuencias	TLD	
	Frames evaluados	Tiempo por *frame* (s)
I_BS_01-60 %	200	0,372
I_BS_02-60 %	0	0
I_CA_01-60 %	125	0,21
I_OC_02-60 %	100	0,037
I_SM_01-60 %	135	0,3
MEDIA		0,184

Tabla C.3: Tiempo por *frame* de los algoritmos de *tracking* en las distintas secuencias.

Secuencias	Boosting		MIL	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01	150	0,419	150	0,776
I_BS_02	100	0,363	100	0,734
I_CA_01	241	0,158	241	0,565
I_OC_02	100	0,369	100	0,631
I_SM_01	217	0,35	217	0,656
MEDIA		0,332		0,672

Secuencias	KCF		MedianFlow	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01	150	0,149	150	0,038
I_BS_02	100	0,115	100	0,038
I_CA_01	225	0,931	225	0,038
I_OC_02	100	0,233	75	0,037
I_SM_01	217	0,379	142	0,038
MEDIA		0,361		0,038

Secuencias	TLD	
	Frames evaluados	Tiempo por *frame* (s)
I_BS_01	150	0,692
I_BS_02	100	0,823
I_CA_01	225	0,309
I_OC_02	120	0,825
I_SM_01	191	1,013
MEDIA		0,732

Tabla C.4: Tiempo por *frame* de los algoritmos de *tracking* con resolución al 60 % (Raspberry).

Secuencias	Boosting		MIL	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01-60 %	200	0,247	200	0,617
I_BS_02-60 %	0	0	0	0
I_CA_01-60 %	100	0,265	100	0,7
I_OC_02-60 %	100	0,245	100	0,579
I_SM_01-60 %	136	0,252	161	0,661
MEDIA		0,202		0,511

Secuencias	KCF		MedianFlow	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01-60 %	175	0,118	200	0,023
I_BS_02-60 %	0	0	0	0
I_CA_01-60 %	75	0,115	100	0,023
I_OC_02-60 %	100	0,112	75	0,023
I_SM_01-60 %	161	0,142	135	0,024
MEDIA		0,097		0,019

Secuencias	TLD	
	Frames evaluados	Tiempo por *frame* (s)
I_BS_01-60 %	200	0,928
I_BS_02-60 %	0	0
I_CA_01-60 %	125	0,546
I_OC_02-60 %	75	0,662
I_SM_01-60 %	158	0,737
MEDIA		0,575

Tabla C.5: Tiempo por *frame* de los algoritmos KCF y MedianFlow en todas las secuencias. (Raspberry).

Secuencias	MedianFlow		KCF	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01	150	0,038	150	0,15
I_BS_02	100	0,039	100	0,115
I_CA_01	225	0,038	225	0,931
I_CA_02	472	0,038	447	0,319
I_IL_01	65	0,038	75	0,273
I_IL_02	300	0,038	300	0,336
I_MB_01	150	0,038	125	0,489
I_MB_02	200	0,038	200	0,46
I_MC_01	100	0,039	100	0,207
I_MC_02	77	0,039	77	0,223
I_OC_01	139	0,038	125	0,145
I_OC_02	75	0,038	100	0,235
I_SI_01	161	0,039	175	0,183
I_SI_02	150	0,038	150	0,153
I_SM_01	142	0,038	217	0,38
I_SM_02	165	0,037	190	0,238
I_SM_03	164	0,038	191	0,256
I_SM_04	162	0,037	132	0,157
I_SM_05	162	0,038	126	0,164
I_SM_06	91	0,038	126	0,157
I_SM_07	109	0,039	137	0,145
I_SM_08	87	0,039	126	0,196
I_SM_09	150	0,038	126	0,155
I_SM_10	86	0,038	133	0,182
I_SM_11	137	0,038	134	0,184
I_SM_12	136	0,038	126	0,201
Media		0,038		0,255

Tabla C.6: Tiempo por *frame* de los algoritmos KCF y MedianFlow en todas las secuencias con resolución al al 60 % (Raspberry).

Secuencias	MedianFlow		KCF	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01	200	0,023	175	0,117
I_BS_02	0	0	0	0
I_CA_01	100	0,023	75	0,115
I_CA_02	0	0	0	0
I_IL_01	75	0,024	75	0,179
I_IL_02	150	0,024	150	0,144
I_MB_01	0	0	0	0
I_MB_02	150	0,024	150	0,117
I_MC_01	50	0,024	50	0,115
I_MC_02	77	0,024	77	0,12
I_OC_01	50	0,024	50	0,115
I_OC_02	75	0,023	100	0,119
I_SI_01	150	0,023	175	0,115
I_SI_02	96	0,023	96	0,156
I_SM_01	135	0,023	161	0,141
I_SM_02	135	0,024	160	0,144
I_SM_03	210	0,023	150	0,148
I_SM_04	143	0,024	143	0,122
I_SM_05	167	0,024	167	0,116
I_SM_06	167	0,024	142	0,116
I_SM_07	143	0,024	143	0,125
I_SM_08	143	0,024	125	0,115
I_SM_09	143	0,024	107	0,147
I_SM_10	143	0,024	143	0,146
I_SM_11	118	0,024	168	0,133
I_SM_12	143	0,024	168	0,153
Media		0,021		0,116

Tabla C.7: Tiempo por *frame* de los algoritmos KCF y MedianFlow en todas las secuencias (PC).

Secuencias	MedianFlow		KCF	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01	150	0,006	150	0,014
I_BS_02	100	0,005	100	0,01
I_CA_01	225	0,005	225	0,077
I_CA_02	460	0,005	447	0,037
I_IL_01	69	0,005	75	0,028
I_IL_02	300	0,005	300	0,036
I_MB_01	150	0,006	125	0,051
I_MB_02	200	0,006	200	0,051
I_MC_01	100	0,006	100	0,024
I_MC_02	77	0,005	77	0,029
I_OC_01	139	0,005	125	0,016
I_OC_02	75	0,005	100	0,028
I_SI_01	175	0,005	175	0,019
I_SI_02	150	0,005	150	0,017
I_SM_01	142	0,005	217	0,037
I_SM_02	165	0,005	190	0,023
I_SM_03	164	0,005	191	0,025
I_SM_04	162	0,005	132	0,019
I_SM_05	162	0,006	126	0,021
I_SM_06	150	0,006	126	0,021
I_SM_07	109	0,006	137	0,02
I_SM_08	87	0,006	126	0,022
I_SM_09	144	0,005	126	0,019
I_SM_10	86	0,006	133	0,022
I_SM_11	137	0,005	134	0,022
I_SM_12	136	0,005	126	0,02
Media		0,005		0,027

Tabla C.8: Tiempo por *frame* de los algoritmos KCF y MedianFlow en todas las secuencias con resolución al 60 % (PC).

Secuencias	MedianFlow		KCF	
	Frames evaluados	Tiempo por *frame* (s)	*Frames* evaluados	Tiempo por *frame* (s)
I_BS_01	200	0,003	175	0,008
I_BS_02	0	0	0	0
I_CA_01	100	0,003	75	0,009
I_CA_02	0	0	0	0
I_IL_01	75	0,003	75	0,011
I_IL_02	150	0,003	150	0,009
I_MB_01	0	0	0	0
I_MB_02	150	0,003	150	0,008
I_MC_01	50	0,004	50	0,008
I_MC_02	77	0,003	77	0,008
I_OC_01	50	0,004	50	0,008
I_OC_02	75	0,003	100	0,007
I_SI_01	150	0,004	175	0,007
I_SI_02	96	0,003	96	0,008
I_SM_01	135	0,003	161	0,009
I_SM_02	135	0,004	160	0,009
I_SM_03	210	0,003	150	0,009
I_SM_04	143	0,003	143	0,008
I_SM_05	167	0,003	167	0,007
I_SM_06	167	0,003	142	0,007
I_SM_07	143	0,003	143	0,008
I_SM_08	143	0,004	125	0,007
I_SM_09	143	0,003	107	0,008
I_SM_10	143	0,003	143	0,009
I_SM_11	118	0,003	168	0,009
I_SM_12	143	0,003	168	0,01
Media		0,003		0,007

Apéndice D

Evaluación de la precisión del *tracking*

En este anexo se detallan los resultados obtenidos al medir la precisión de los algoritmos de *tracking* sobre las diferentes secuencias de LASIESTA. Los resultados mostrados en las tablas se han calculado usando las expresiones (3.2) y (3.3). En cada fila se pueden observar la secuencia sobre la que se ha aplicado el algoritmo, la cantidad de imágenes que han sido evaluadas, la cantidad de imágenes que se considera que han sido evaluadas correctamente y la precisión del algoritmo para esa secuencia. Adicionalmente, también se muestra la precisión media del algoritmo. Las características de las diferentes secuencias se pueden encontrar en la Tabla B.2.

A continuación se describen los datos de cada tabla:

- La Tabla D.1 muestra la precisión de todos los algoritmos de *tracking* valorados sobre una selección de secuencias de LASIESTA, cuyos *frames* han sido usados con su resolución original (352x288).
- La Tabla D.2 muestra la precisión de todos los algoritmos de *tracking* valorados sobre una selección de secuencias de LASIESTA. La resolución de las imágenes de estas secuencias ha sido reducida al 60 %.
- En las Tablas D.3 y D.4 se muestra la precisión del algoritmo KCF sobre todas las secuencias de interior de LASIESTA con resolución original y resolución al 60 %, respectivamente.
- En las Tablas D.5 y D.6 se muestra la precisión del algoritmo MedianFlow sobre todas las secuencias de interior de LASIESTA con resolución original y resolución al 60 %, respectivamente.

Para comprobar si la inicialización de los algoritmos ha sido buena también se dispone de la precisión del detector. Este dato ha sido calculado usando la expresión (3.4). En cada fila de la tabla se puede ver la secuencia con la que se han calculado los datos y la precisión del detector en esa secuencia. En la Tabla D.7 se puede ver la precisión del detector en todas las secuencias con resolución original y resolución al 60 %. Se muestran los resultados comunes con todos los algoritmos de *tracking*, con los que se obtuvo una buena precisión en casi todas

las secuencias, apenas variando la precisión dle detector al usar un algoritmo u otro.

Tabla D.1: Precisión de los algoritmos de *tracking* en el subconjunto de secuencias de evaluación inicial.

Secuencias	Boosting		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01	150	123	0,82
I_BS_02	100	42	0,42
I_CA_01	241	203	0,84
I_OC_02	100	52	0,52
I_SM_01	217	33	0,15
MEDIA			0,55

Secuencias	MIL		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01	150	143	0,95
I_BS_02	100	45	0,45
I_CA_01	241	213	0,88
I_OC_02	100	9	0,09
I_SM_01	217	15	0,07
MEDIA			0,49

Secuencias	KCF		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01	150	145	0,97
I_BS_02	100	75	0,75
I_CA_01	225	225	1
I_OC_02	100	45	0,45
I_SM_01	141	35	0,16
MEDIA			0,67

Secuencias	MedianFlow		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01	150	121	0,81
I_BS_02	100	29	0,29
I_CA_01	225	170	0,76
I_OC_02	75	8	0,11
I_SM_01	142	10	0,07
MEDIA			0,41

Secuencias	TLD		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01	150	96	0,64
I_BS_02	100	23	0,23
I_CA_01	225	122	0,54
I_OC_02	100	8	0,07
I_SM_01	141	16	0,08
MEDIA			0,31

Tabla D.2: Precisión de los algoritmos de *tracking* en el subconjunto de secuencias de evaluación inicial con resolución al 60 %.

Secuencias	Boosting		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01-60 %	200	22	0,11
I_BS_02-60 %	0	0	0
I_CA_01-60 %	100	68	0,68
I_OC_02-60 %	100	50	0,5
I_SM_01-60 %	136	23	0,17
MEDIA			0,29

Secuencias	MIL		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01-60 %	200	23	0,12
I_BS_02-60 %	0	0	0
I_CA_01-60 %	100	94	0,94
I_OC_02-60 %	100	11	0,11
I_SM_01-60 %	161	28	0,17
MEDIA			0,27

Secuencias	KCF		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01-60 %	175	25	0,14
I_BS_02-60 %	0	0	0
I_CA_01-60 %	75	75	1
I_OC_02-60 %	100	50	0,5
I_SM_01-60 %	161	5	0,03
MEDIA			0,33

Secuencias	MedianFlow		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01-60 %	200	20	0,1
I_BS_02-60 %	0	0	0
I_CA_01-60 %	100	95	0,95
I_OC_02-60 %	75	11	0,15
I_SM_01-60 %	135	61	0,45
MEDIA			0,33

Secuencias	TLD		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01-60 %	200	24	0,12
I_BS_02-60 %	0	0	0
I_CA_01-60 %	125	39	0,31
I_OC_02-60 %	75	1	0,01
I_SM_01-60 %	158	4	0,03
MEDIA			0,09

Tabla D.3: Precisión del algoritmo KCF en todas las secuencias.

Secuencias	KCF		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01	150	145	0,97
I_BS_02	100	75	0,75
I_CA_01	225	225	1
I_CA_02	447	225	0,50
I_IL_01	75	52	0,69
I_IL_02	300	280	0,93
I_MB_01	125	50	0,4
I_MB_02	200	93	0,47
I_MC_01	100	51	0,51
I_MC_02	77	41	0,53
I_OC_01	125	85	0,68
I_OC_02	100	45	0,45
I_SI_01	175	15	0,09
I_SI_02	150	140	0,93
I_SM_01	217	35	0,16
I_SM_02	190	0	0
I_SM_03	191	0	0
I_SM_04	132	30	0,23
I_SM_05	126	10	0,08
I_SM_06	126	3	0,02
I_SM_07	137	7	0,05
I_SM_08	126	33	0,26
I_SM_09	126	10	0,08
I_SM_10	133	25	0,19
I_SM_11	134	27	0,20
I_SM_12	126	16	0,13
Media			0,38

Tabla D.4: Precisión del algoritmo KCF en todas las secuencias (resolución al 60 %).

Secuencias	KCF		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01	175	25	0,14
I_BS_02	0	0	0
I_CA_01	75	75	1
I_CA_02	0	0	0
I_IL_01	75	56	0,75
I_IL_02	150	141	0,94
I_MB_01	0	0	0
I_MB_02	150	150	1
I_MC_01	50	50	1
I_MC_02	77	42	0,55
I_OC_01	50	27	0,54
I_OC_02	100	56	0,56
I_SI_01	175	108	0,62
I_SI_02	96	77	0,80
I_SM_01	161	5	0,03
I_SM_02	160	23	0,14
I_SM_03	150	58	0,39
I_SM_04	143	38	0,27
I_SM_05	167	42	0,25
I_SM_06	142	31	0,22
I_SM_07	143	66	0,46
I_SM_08	125	53	0,42
I_SM_09	107	37	0,35
I_SM_10	143	33	0,23
I_SM_11	168	52	0,31
I_SM_12	168	90	0,54
Media			0,44

Tabla D.5: Precisión del algoritmo MedianFlow en todas las secuencias.

Secuencias	MedianFlow		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01	150	121	0,81
I_BS_02	100	29	0,29
I_CA_01	225	170	0,76
I_CA_02	472	219	0,46
I_IL_01	65	46	0,71
I_IL_02	300	167	0,56
I_MB_01	150	36	0,24
I_MB_02	200	75	0,38
I_MC_01	100	20	0,2
I_MC_02	77	21	0,27
I_OC_01	139	34	0,24
I_OC_02	75	8	0,11
I_SI_01	161	39	0,24
I_SI_02	150	117	0,78
I_SM_01	142	10	0,07
I_SM_02	165	24	0,15
I_SM_03	164	31	0,19
I_SM_04	162	37	0,23
I_SM_05	162	9	0,06
I_SM_06	91	14	0,15
I_SM_07	109	23	0,21
I_SM_08	87	25	0,29
I_SM_09	150	50	0,33
I_SM_10	86	38	0,44
I_SM_11	137	27	0,2
I_SM_12	136	19	0,14
Media			0,33

Tabla D.6: Precisión del algoritmo MedianFlow en todas las secuencias (resolución al 60 %).

Secuencias	MedianFlow		
	Frames evaluados	*Frames* correctos (umbral 0,8)	Precisión
I_BS_01	200	20	0,1
I_BS_02	0	0	0
I_CA_01	100	95	0,95
I_CA_02	0	0	0
I_IL_01	75	66	0,88
I_IL_02	150	87	0,58
I_MB_01	0	0	0
I_MB_02	150	134	0,89
I_MC_01	50	22	0,44
I_MC_02	77	32	0,42
I_OC_01	50	18	0,36
I_OC_02	75	11	0,15
I_SI_01	150	33	0,22
I_SI_02	96	64	0,67
I_SM_01	135	61	0,45
I_SM_02	135	54	0,4
I_SM_03	210	79	0,38
I_SM_04	143	47	0,33
I_SM_05	167	46	0,28
I_SM_06	167	52	0,31
I_SM_07	143	55	0,38
I_SM_08	143	46	0,32
I_SM_09	143	30	0,21
I_SM_10	143	53	0,37
I_SM_11	118	53	0,45
I_SM_12	143	62	0,43
Media			0,38

Tabla D.7: Precisión del detector sobre todas las secuencias con resolución original y al 60 %.

Secuencias	Precisión	Precisión (resolución 60 %)
I_BS_01	1	0,2
I_BS_02	1	0
I_CA_01	1	1
I_CA_02	0,055	0
I_IL_01	1	1
I_IL_02	1	1
I_MB_01	1	0
I_MB_02	1	1
I_MC_01	1	1
I_MC_02	1	1
I_OC_01	1	1
I_OC_02	1	1
I_SI_01	1	1
I_SI_02	1	1
I_SM_01	0,67	1
I_SM_02	0,75	1
I_SM_03	0,67	1
I_SM_04	1	1
I_SM_05	1	1
I_SM_06	1	1
I_SM_07	1	1
I_SM_08	1	1
I_SM_09	1	1
I_SM_10	1	1
I_SM_11	1	1
I_SM_12	1	1

Apéndice E

Manual de uso

E.1. Ejecución

Para ejecutar el sistema (Figura E.1) es necesario tener los ficheros *vigilancia.py*, *deteccion.py*, *seguimiento.py* y *controlMotor.py* en la misma ubicación. Para la ejecución por defecto bastará con ejecutar el comando *python vigilancia.py* (Figura E.2). Adicionalmente se pueden usar varios parámetros para modificar el comportamiento del sistema, se puede observar un ejemplo de este tipo de ejecución en la Figura E.3. A continuación se detallan los parámetros que el sistema puede utilizar:

- **-w.** Se utiliza la *webcam* de un ordenador, en lugar de utilizar la cámara del prototipo, para obtener la secuencia de vídeo. Esta opción esta ideada para utilizarla en pruebas. Esta opción desactivará el uso del motor.
- **-d [ruta a secuencia de frames].** Analiza *frames* ya guardados en el sistema en lugar de en tiempo real. La ruta indicada debe contener los *frames* a analizar. Esta opción desactivará el uso del motor.
- **-t [algoritmo].** Indica el algoritmo de *tracking* a utilizar. El algoritmo por defecto es MedianFlow.
- **-db [ruta a clave de dropbox].** Indica la ruta en la que se encuentra el fichero con la clave para subir las imágenes a la cuenta de dropbox[1]. Si este parámetro no se utiliza las imágenes no serán subidas a dropbox.
- **-e [extension].** Formato de imágen en el que se guardaran los *frames* subidos a dropbox. Admite los formatos *png*, *jpg*, *ppm*, *pbm* y *pgm*. El formato utilizado por defecto es *jpg*.
- **-r [ruta de dropbox].** Indica la ruta en la que se guardaran las imágenes subidas a dropbox. Por defecto se guardaran en */raspberry_vigilancia/*.
- **-res [resosución].** Resolución de las imágenes capturadas por la cámara. Por defecto se usa la resolución 320x240.

[1] `https://www.dropbox.com`

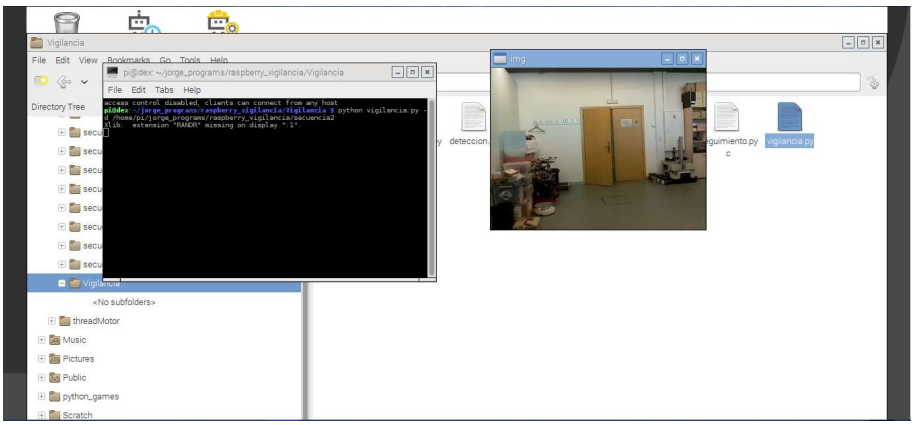


Figura E.1: Imagen completa del sistema en ejecución.

- **-nm.** Este *flag* hace que no se use el motor durante la ejecución del programa.
- **-f [FPS].** *Frames* por segundo que debe capturar la cámara del prototipo. El valor por defecto es 32 *Frames* por segundo.

E.2. Dropbox

Para poder subir las imágenes detectadas a dropbox, se necesita una clave que permita a la aplicación acceder a la cuenta deseada para subir las imágenes. Para obtener esta clave se debe acceder a la página de creación de *apps* de dropbox (Figura E.4) y rellenar los campos requeridos.

Hecho esto aparecerá una página con la configuración de la aplicación creada. En esta página se debe generar la clave a usar por el sistema, para ello hay que pulsar el botón *Generate* (Figura E.5). Una vez que aparezca la clave, solo habrá que copiarla en un fichero, cuya ruta será la que se indique al sistema tal y como se explica en la Sección E.1.

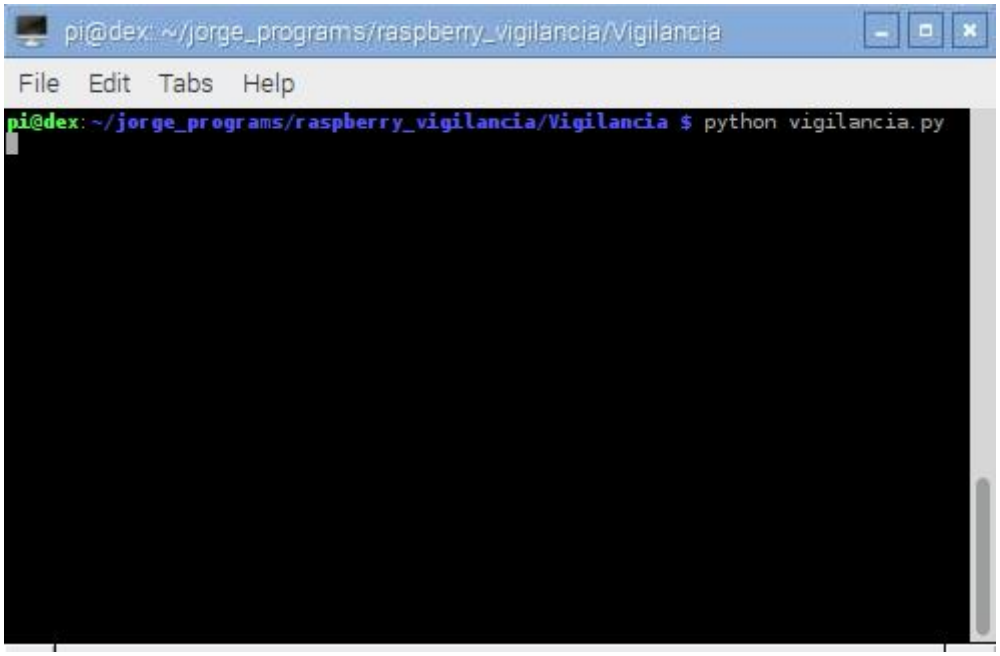


Figura E.2: Ejecución por defecto del programa.

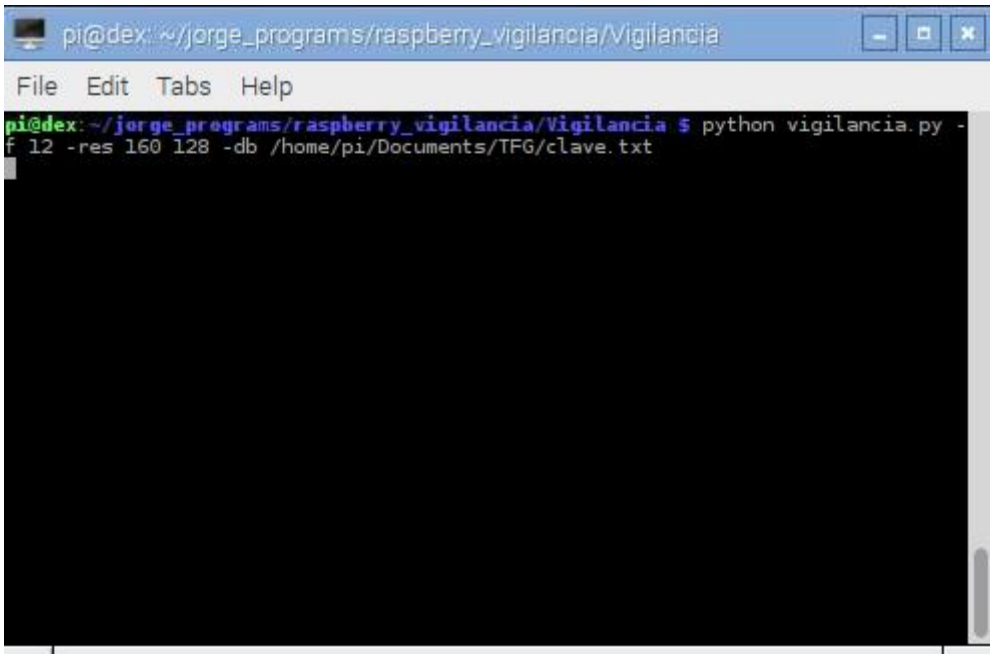


Figura E.3: Ejecución del programa con parámetros. En este ejemplo el programa realiza *streaming* a 12 *Frames* por segundo, los *frames* tienen una resolución de 160x128 y se indica que se subirán imágenes a dropbox indicando la ruta a la clave.

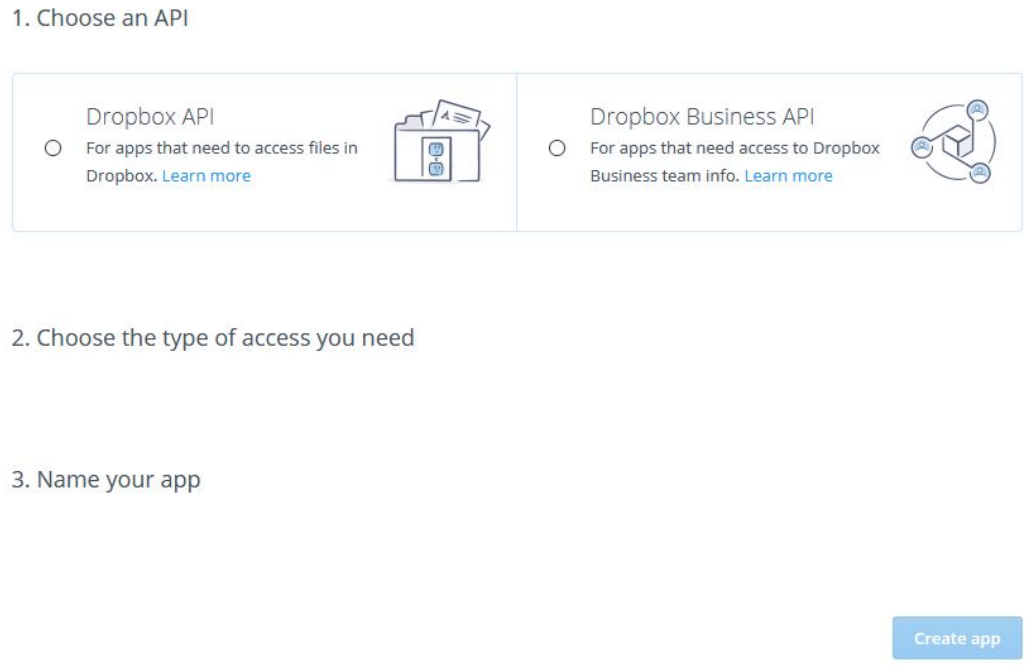


Figura E.4: Página de creación de *apps* de dropbox.

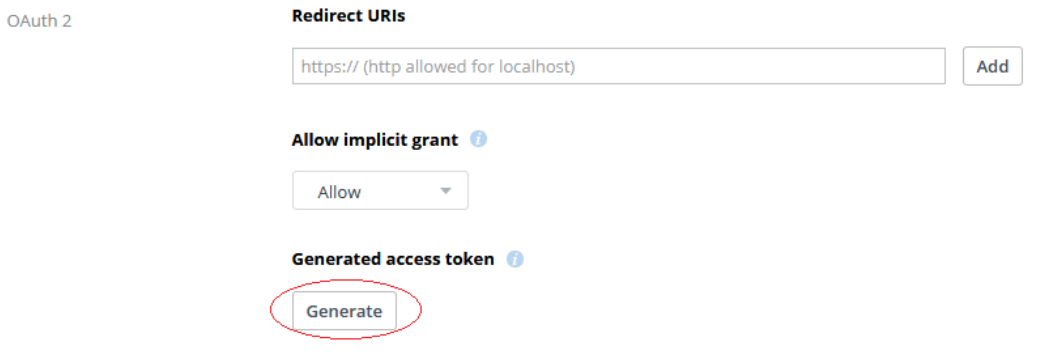


Figura E.5: Botón para generar la clave a usar en el sistema.