UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

GoBees

Monitorización del estado de una colmena mediante la cámara de un smartphone.

Presentado por David Miguel Lozano
en Universidad de Burgos — 28 de enero de 2017
Tutores: José Francisco Díez Pastor
y Raúl Marticorena Sánchez

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. David Miguel Lozano, con DNI dni, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 28 de enero de 2017

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Introducción

Las abejas son una pieza clave en nuestro ecosistema. La producción de alimentos a nivel mundial y la biodiversidad de nuestro planeta dependen en gran medida de ellas. Son las encargadas de polinizar el 70 % de los cultivos de comida, que suponen un 90 % de la alimentación humana [13]. Sin embargo, la población mundial de abejas está disminuyendo a pasos agigantados en los últimos años debido, entre otras causas, al uso extendido de plaguicidas tóxicos, parásitos como la varroa o la expansión del avispón asiático [16].

Actualmente los apicultores inspeccionan sus colmenares de forma manual. Uno de los indicadores que más información les proporciona es la actividad de vuelo de la colmena (número de abejas en vuelo a la entrada de la colmena en un determinado instante) [2]. Este dato, junto con información previa de la colmena y conocimiento de las condiciones locales, permite al apicultor conocer el estado de la colmena con bastante seguridad, pudiendo determinar si esta necesita o no una intervención.

La actividad de vuelo de una colmena varía dependiendo de múltiples factores, tanto externos como internos a la colmena. Entre ellos se encuentran la propia población de la colmena, las condiciones meteorológicas, la presencia de enfermedades, parásitos o depredadores, la exposición a tóxicos, la presencia de fuentes de néctar, etc. A pesar de los numerosos factores que influyen en la actividad de vuelo, su conocimiento es de gran ayuda para la toma de decisiones por parte del apicultor. Ya que este posee información sobre la mayoría de los factores necesarios para su interpretación.

La captación prolongada de la actividad de vuelo de forma manual es muy costosa, tediosa y puede introducir una tasa de error elevada. Es por esto que a lo largo de los años se haya intentado automatizar este proceso de muy diversas maneras. Los primeros intentos se remontan a principios del siglo XX, donde se desarrollaron contadores por contacto eléctrico [24]. Otros métodos posteriores se basan en sensores de infrarrojos [34], sensores capacitivos [3], códigos de barras [6] o incluso en microchips acoplados a las abejas [7]. En

los últimos años, se han desarrollado numerosos métodos basados en visión artificial [2, 4, 5, 35].

Los métodos basados en contacto, sensores de infrarrojos o capacitivos tienen el inconveniente de que es necesario realizar modificaciones en la colmena, mientras que en los basados en códigos de barras o microchips es necesario manipular las abejas directamente. Estos motivos los convierten en métodos poco prácticos fuera del campo investigador. Por el contrario, la visión artificial aporta un gran potencial, ya que evita tener que realizar ningún tipo de modificación ni en el entorno, ni en las abejas. Además, abre la puerta a la monitorización de nuevos parámetros como la detección de enjambrazón (división de la colmena y salida de un enjambre) o la detección de amenazas (avispones, abejaruco, etc.).

Todos los métodos basados en visión artificial propuestos hasta la fecha utilizan hardware específico con un coste elevado. En este trabajo se propone un método de monitorización de la actividad de vuelo de una colmena mediante la cámara de un *smartphone* con Android.

El método propuesto podría revolucionar el campo de la monitorización de la actividad de vuelo de colmenas, ya que lo hace accesible al gran público. Ya no es necesario contar con costoso hardware, difícil de instalar. Solamente es necesario disponer de un *smartphone* con cámara y la aplicación GoBees. Además, esta facilita la interpretación de los datos, representándolos gráficamente y añadiendo información adicional como la situación meteorológica. Permitiendo a los apicultores centrar su atención donde realmente es necesaria.

1.1. Estructura de la memoria

La memoria sigue la siguiente estructura:

- **Introducción:** breve descripción del problema a resolver y la solución propuesta. Estructura de la memoria y listado de materiales adjuntos.
- **Objetivos del proyecto:** exposición de los objetivos que persigue el proyecto.
- **Conceptos teóricos:** breve explicación de los conceptos teóricos clave para la comprensión de la solución propuesta.
- **Técnicas y herramientas:** listado de técnicas metodológicas y herramientas utilizadas para gestión y desarrollo del proyecto.
- **Aspectos relevantes del desarrollo:** exposición de aspectos destacables que tuvieron lugar durante la realización del proyecto.
- **Trabajos relacionados:** estado del arte en el campo de la monitorización de la actividad de vuelo de colmenas y proyectos relacionados.

- **Conclusiones y líneas de trabajo futuras:** conclusiones obtenidas tras la realización del proyecto y posibilidades de mejora o expansión de la solución aportada.

Junto a la memoria se proporcionan los siguientes anexos:

- **Plan del proyecto software:** planificación temporal y estudio de viabilidad del proyecto.
- **Especificación de requisitos del software:** se describe la fase de análisis; los objetivos generales, el catálogo de requisitos del sistema y la especificación de requisitos funcionales y no funcionales.
- **Especificación de diseño:** se describe la fase de diseño; el ámbito del software, el diseño de datos, el diseño procedimental y el diseño arquitectónico.
- **Manual del programador:** recoge los aspectos más relevantes relacionados con el código fuente (estructura, compilación, instalación, ejecución, pruebas, etc.).
- **Manual de usuario:** guía de usuario para el correcto manejo de la aplicación.

1.2. Materiales adjuntos

Los materiales que se adjuntan con la memoria son:

- Aplicación para Android GoBees.
- Aplicación Java para el desarrollo del algoritmo.
- Aplicación Java para el etiquetado de fotogramas.
- JavaDoc.
- *Dataset* de vídeos de prueba.

Además, los siguientes recursos están accesibles a través de internet:

- Página web del proyecto [23].
- GoBees en Google Play [20].
- Repositorio del proyecto [22].
- Repositorio de las herramientas desarrolladas para el proyecto [21].

Objetivos del proyecto

A continuación, se detallan los diferentes objetivos que han motivado la realización del proyecto.

2.1. Objetivos generales

- Desarrollar una aplicación para *smartphone* que permita la monitorización de la actividad de vuelo de una colmena a través de su cámara.
- Facilitar la interpretación de los datos recogidos mediante representaciones gráficas.
- Aportar información extra a los datos de actividad que ayude en la toma de decisiones.
- Almacenar todos los datos generados de forma estructurada y fácilmente accesible.

2.2. Objetivos técnicos

- Desarrollar un algoritmo de visión artificial con OpenCV que permita contar el número de abejas en cada fotograma en tiempo real.
- Desarrollar una aplicación Android con soporte para API 19 y superiores.
- Aplicar la arquitectura MVP (*Model-View-Presenter*) en el desarrollo de la aplicación.
- Utilizar Gradle como herramienta para automatizar el proceso de construcción de software.
- Utilizar Git como sistema de control de versiones distribuido junto con la plataforma GitHub.
- Hacer uso de herramientas de integración continua como Travis, Codecov, Code Climate, SonarQube o VersionEye en el repositorio.

- Aplicar la metodología ágil Scrum junto con TDD (*Test Driven Development*) en el desarrollo del software.
- Realizar test unitarios, de integración y de interfaz.
- Utilizar ZenHub como herramienta de gestión de proyectos.
- Utilizar un sistema de documentación continua como Read the Docs.
- Distribuir la aplicación resultante en la plataforma Google Play.
- Realizar una página web para la difusión de la aplicación.

2.3. Objetivos personales

- Realizar una aportación a la modernización de la apicultura.
- Abarcar el máximo número de conocimientos adquiridos durante la carrera.
- Explorar metodologías y herramientas novedosas utilizadas en el mercado laboral.
- Adentrarme en el campo de la visión artificial.
- Profundizar en el desarrollo de aplicaciones Android.

Conceptos teóricos

La parte del proyecto con mayor complejidad teórica radica en el algoritmo de visión artificial, el cual se puede dividir en cuatro fases. En primer lugar, se realiza un preprocesado de la señal de entrada para optimizarla. En segundo lugar, se substrae el fondo para segmentar los objetos en movimiento. Posteriormente, se realiza un posprocesado de la señal para mejorar los resultados obtenidos de la substracción del fondo. Y por último, se clasifican y cuentan los contornos que pueden pertenecer a una abeja.

A continuación se exponen los conceptos teóricos que conlleva cada fase.

3.1. Preprocesado

Antes de aplicar el algoritmo de substracción del fondo, es recomendable realizar un preprocesado de los fotogramas para facilitar el procesado posterior, minimizar el ruido y optimizar los resultados. A continuación se explican las técnicas utilizadas.

Conversión de RGB a escala de grises

Los fotogramas captados por la cámara se devuelven en formato RGB. En este formato se poseen tres matrices, una por cada canal (*Red-Green-Blue*). La suma aditiva de los tres canales resulta en una imagen a color.

Sin embargo, el color no proporciona ninguna información relevante en nuestra tarea de identificación de abejas. Es por esto que se pueden convertir los fotogramas de RGB a escala de grises. De esta manera, se trabajará solamente con una matriz de píxeles en lugar de tres. Simplificando, en gran medida, el número de operaciones a realizar y por tanto, aumentado el rendimiento de nuestro algoritmo final.

OpenCV utiliza la conversión colométrica a escala de grises [30]. Esta técnica se basa en principios colométricos para ajustar la luminancia de la imagen a

color y la imagen resultante. Devolviendo una imagen con la misma luminancia absoluta y la misma percepción de luminosidad [38].

Utiliza la siguiente fórmula para calcular la luminancia resultante:

$$\text{RGB[A] to Gray:} \quad Y \leftarrow 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B$$

La fórmula calcula la luminancia de una forma no lineal, sin necesidad de realizar una expansión gamma. Los coeficientes intentan imitar la percepción de intensidad medida por un humano tricromático, siendo más sensible al color verde y menos al color azul [38].

Desenfoque Gaussiano

Las imágenes captadas pueden contener ruido que puede dificultar su procesamiento. El ruido son variaciones aleatorias del brillo o del color de una imagen. Una técnica que permite reducirlo es el desenfoque.

En nuestro caso hemos utilizado desenfoque Gaussiano, un filtro de paso bajo que reduce las componentes de alta frecuencia de la imagen utilizando para ello una convolución con una función Gaussiana [37]. Se diferencia del desenfoque promedio en que da más peso a los vecinos cercanos, siendo estos más influyentes en el resultado.

El kernel utilizado en la convolución es una muestra discreta de una función Gaussiana. En nuestro caso utilizamos un kernel 3x3 que se corresponde con: [17]

$$M = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

En la siguiente imagen se puede ver el resultado de aplicar esta fase a la imagen de entrada:

Figura 3.1: Resultado de la fase de preprocesado.

3.2. Substracción del fondo

En un sistema de monitorización por vídeo resulta de gran interés el poder extraer los objetos en movimiento del resto de la imagen. Esto se conoce como extracción del fondo, en inglés *background subtraction* o *foreground detection*, y consiste en clasificar todos los píxeles de un determinado fotograma bien como fondo, o como primer plano [36]. En primer plano se engloban todos los objetos en movimiento, mientras que en el fondo se encuentran todos los objetos estáticos junto con posibles sombras, cambios de iluminación u otros objetos en movimiento que no son de interés, como puede ser la rama de un árbol balanceándose por el viento. Se trata de un paso crítico, ya que algoritmos posteriores dependen en gran medida de los resultados de este.

En nuestro proyecto, la toma de imágenes se realiza mediante una cámara estática. Esto facilita en parte la detección del fondo, ya que este se corresponderá con todos los píxeles estáticos. Sin embargo, como trabajamos en un entorno al aire libre, tenemos que lidiar también con cambios de iluminación, sombras, u otros objetos móviles que no son de nuestro interés (falsos positivos).

Con OpenCV para Android podemos implementar varios algoritmos básicos de extracción del fondo y además, nos proporciona la implementación de dos algoritmos más sofisticados: `BackgroundSubtractorMOG` y `BackgroundSubtractorKNN`.

Tras realizar un estudio de todos ellos, nos decantamos finalmente por `BackgroundSubtractorMOG2`. A continuación, explicamos el funcionamiento de todos los algoritmos que se probaron, así como los resultados que proporcionaron.

Substracción con imagen de referencia

Se parte de una imagen de referencia del fondo, en la que no haya ningún objeto en movimiento. A partir de esta, se obtienen los elementos en movimiento substrayendo a cada fotograma la imagen tomada como referencia.

Este método, al tomar un modelo del fondo tan sencillo y estático, es muy vulnerable a cambios en la escena (iluminación, sombras, objetos del fondo con ligeros movimientos, pequeñas oscilaciones de la cámara, etc.). Sin embargo, ofrece muy buenos resultados cuando se trabaja en una escena con la iluminación y los elementos controlados, ya que al ser tan simple, es muy eficiente [8].

Para implementar este algoritmo con OpenCV, se hace uso de la función `Core.absdiff()`.

En nuestro problema, al trabajar al aire libre nos es imposible utilizar este algoritmo. Ya que el modelo del fondo cambia constantemente.

Substracción del fotograma anterior

En este método, el modelo del fondo se extrae del fotograma anterior. De tal manera, que a cada nuevo fotograma se le substrae el anterior.

De esta manera se mejora la respuesta a cambios en la escena, como los cambios de iluminación. Sin embargo, si un objeto en movimiento se queda estático en la imagen, este deja de ser detectado [1].

La implementación se realiza como en la técnica anterior, variando el modelo del fondo.

Tras probarlo en nuestro problema específico, vimos que no nos era de utilidad. Ya que el fotograma resultante de la diferencia contenía las abejas por duplicado (la abeja en el fotograma actual y misma abeja en el fotograma anterior en otra posición).

Substracción del acumulado de los fotogramas anteriores

Una mejora interesante del algoritmo anterior supone tomar como modelo del fondo un acumulado de los fotogramas anteriores de acuerdo a un ratio de aprendizaje. De esta forma, se puede lidiar con cambios en el fondo de la imagen dinámicamente. El modelo se calcula de acuerdo a la siguiente fórmula:

$$u_t = (1 - \alpha)u_{t-1} + \alpha\ p_t$$

Donde p_t es el nuevo valor del píxel, u_{t-1} es la media del fondo en el instante $t - 1$, u_t es la nueva media del fondo y α es el ratio de aprendizaje (cómo de rápido olvida los *frames* anteriores) [1].

OpenCV provee la función `Imgproc.accumulateWeighted()` que implementa la fórmula anterior por nosotros. Haciendo uso de esta función y de la utilizada en la sección anterior podemos implementar este algoritmo.

Tras probarlo, vimos que tenía una eficiencia muy buena y se adaptaba a los cambios correctamente. Sin embargo, no era capaz de diferenciar las sombras de las abejas, por lo que se obtenían falsos positivos.

BackgroundSubtractorKNN

Se trata de un método que se basa en el algoritmo de clasificación supervisada *K Nearest Neighbors* (k-nn). El algoritmo fue propuesto en el artículo [43]. Y de acuerdo con sus conclusiones, es muy eficiente cuando el número de píxeles que se corresponden con el primer plano es bajo.

La clase de OpenCV que lo implementa es `BackgroundSubtractorKNN`. Los parámetros más importantes son:

- `history`: número de fotogramas recientes que afectan al modelo del fondo.
- `dist2Threshold`: umbral de la distancia al cuadrado entre el píxel y la muestra para decidir si un píxel está cerca de esa muestra.
- `detectShadows`: con un valor verdadero detecta las sombras (aumenta considerablemente el tiempo de procesado).

En nuestras pruebas, el algoritmo proporcionaba unos resultados buenos pero su tiempo de ejecución era muy elevado (entorno a 25ms/frame). Como el tiempo de ejecución es un factor clave en nuestro proyecto, se descartó el uso de este algoritmo.

BackgroundSubtractorMOG2

`BackgroundSubtractorMOG2` es una mejora del algoritmo `BackgroundSubtractorMOG`. En la versión original de OpenCV se encuentran implementados ambos, sin embargo, en los *wrappers* para Android solo disponemos de la revisión.

`BackgroundSubtractorMOG` está basado en el modelo *Gaussian Mixture* (GMM). Se trata de un modelo compuesto por la suma de varias distribuciones Gaussianas que, correctamente elegidas, permiten modelar cualquier distribución [26].

El algoritmo de substracción del fondo fue propuesto en el artículo [41] y modela cada píxel del fondo como la mezcla de K distribuciones Gaussianas. Los pesos de la mezcla representan las proporciones de tiempo que el color de ese píxel se ha mantenido en la escena. Siendo los colores de fondo más probables los que más permanezcan y sean más estáticos [29].

`BackgroundSubtractorMOG2` se basa en los mismos principios que su antecesor pero implementa una mejora sustancial. Es el propio algoritmo el que selecciona el número adecuado de distribuciones Gaussianas necesarias para modelar cada píxel. De esta manera, se mejora notablemente la adaptabilidad del algoritmo a variaciones en la escena. Fue propuesto en los artículos [42] y [43].

El código fuente de este algoritmo está disponible en [27] (interfaz) y [28] (implementación).

La clase de OpenCV que lo implementa es `BackgroundSubtractorMOG2`. Posee los siguientes parámetros configurables: [33]

- `history`: número de fotogramas recientes que afectan al modelo del fondo. Se representa en la literatura como `T`. Por defecto, 500 fotogramas. Nosotros hemos obtenido buenos resultados con valores de entorno a 50.

- `learningRate`: valor entre 0 y 1 que indica como de rápido aprende el modelo. Si se establece un valor de -1 el algoritmo elige automáticamente el ratio. 0 significa que el modelo del fondo no se actualiza para nada, mientras que 1 supone que el modelo del fondo se reinicializa completamente cada nuevo fotograma. En la literatura podemos encontrar este parámetro como `alfa`. Si el intervalo que se quiere considerar es `history`, se debe establecer `alfa=1/history` (valor por defecto). También se pueden mejorar los resultados iniciales estableciendo `alfa=1` en el instante 0 e ir decrementándolo hasta `alfa=1/history`. De esta manera, en el inicio aprende rápidamente, pero una vez estabilizada la situación las variaciones afectan menos al modelo. En nuestro caso, el valor por defecto ha funcionado correctamente.
- `backgroundRatio`: si un pixel del primer plano permanece con un valor semi-constante durante `backgroundRatio*history` fotogramas, es considerado fondo y se añade al modelo del fondo como centro de una nueva componente Gaussiana. En los artículos se hace referencia a este parámetro como `TB`. `TB=0.9` es el valor por defecto. Este parámetro nos permite decidir cuando dejar de contar una abeja que se ha quedado inmóvil o un objeto nuevo en la escena como podría ser una hoja que se acaba de caer de un árbol.
- `detectShadows`: con un valor verdadero (valor por defecto) detecta las sombras (aumenta ligeramente el tiempo de procesado). Nos permite despreciar las sombras de las abejas con muy buenos resultados.
- `shadowThreshold`: el algoritmo detecta las sombras comprobando si un píxel es una versión oscurecida del fondo. Este parámetro define cómo de oscura puede ser la sombra como máximo. Por ejemplo, un valor de 0.5 (valor por defecto) significa que si un píxel es más del doble de oscuro, entonces no se considerará sombra. En los artículos se representa como `Tau`.
- `shadowValue`: es el valor utilizado para marcar los píxeles de sombras en la máscara resultante. El valor por defecto es 127. En la máscara devuelta, un valor de 0 siempre se corresponde con un pixel del fondo, mientras que un valor de 255 con un píxel del primer plano.
- `nMixtures`: número máximo de componentes Gaussianas para modelar el modelo del fondo. El número actual se determina dinámicamente para cada píxel. Hemos utilizado el valor por defecto, 5.
- `varThreshold`: umbral utilizado en el cálculo de la distancia cuadrada de Mahalanobis entre el píxel y el modelo del fondo para decidir si una muestra está bien descrita por el modelo o no. Este parámetro no afecta a la actualización del modelo del fondo. Se representa como `Cthr`. Por defecto, 16. Se han obtenido mejores resultados con valores de entorno a 40.
- `varThresholdGen`: umbral sobre la distancia cuadrada de Mahalanobis entre el píxel y el modelo para ayudar a decidir si un píxel está cercano

a alguna de las componentes del modelo. Si no es así, es considerado como primer plano o añadido como centro de una nueva componente (dependiendo del `backgroundRatio`). Se representa como `Tg` y su valor por defecto es 9. Un valor menor genera más componentes Gaussianas, mientras que un valor mayor genera menos.

- `complexityReductionThreshold`: este parámetro define el número de muestras necesarias para probar que una componente existe. Se representa como `CT`. Su valor por defecto es `CT=0.05`. Si se establece su valor a 0 se obtiene un algoritmo similar al de Stauffer & Grimson (no se reduce el número de componentes).
- `varInit`: varianza inicial de cada componente Gaussiana. Afecta a la velocidad de adaptación. Se debe ajustar teniendo en cuenta la desviación estandar de las imágenes. Por defecto es 15.
- `varMin`: varianza mínima. Por defecto, 4.
- `varMax`: varianza máxima. Por defecto, `5*varInit`.

De todos ellos, los parámetros más importantes a ajustar son `history` o `learningRate`, `varThreshold` y `detectShadows`.

La parametrización correcta de este algoritmo es clave para su buen funcionamiento. Por ello, durante las pruebas se integró en nuestra aplicación de desarrollo, permitiendo variar todos estos parámetros en tiempo real. De esta manera, se pudo elegir la mejor configuración para nuestro problema concreto.

En la siguiente imagen se puede ver una captura de nuestra plataforma de desarrollo en la pestaña correspondiente a esta fase:

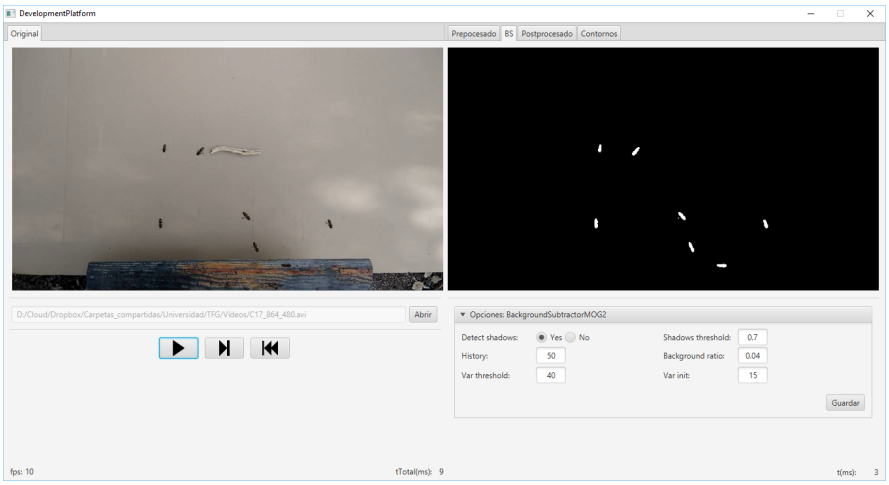

Figura 3.2: Plataforma de desarrollo del algoritmo.

Una vez parametrizado correctamente, vimos como este algoritmo era el que mejores resultados nos proporcionaba. Con un tiempo de ejecución en

nuestro equipo de pruebas de entorno a 4 ms/frame, mucho menor que el proporcionado por `BackgroundSubtractorKNN`, de entorno a 25 ms/frame. El algoritmo detectaba correctamente las abejas, era resistente al ruido, y además, era capaz de diferenciar una abeja de su sombra. Por todos estos motivos, se seleccionó para la fase de substracción del fondo.

En la siguiente imagen se puede ver el resultado de aplicar `BackgroundSubtractorMOG2` a la salida de la fase anterior:

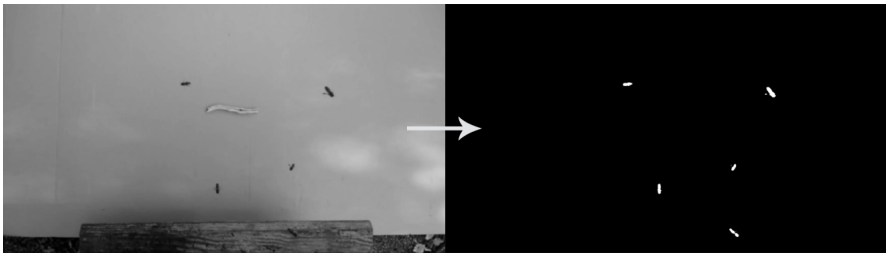

Figura 3.3: Resultado de la fase de substracción del fondo.

Se puede apreciar como ha descartado correctamente las sombras en movimiento de los árboles y se ha quedado únicamente con los objetos en movimiento.

Otros algoritmos

La implementación original de OpenCV implementa otros dos algoritmos más que no están disponibles a través de los *wrappers* de Android.

- `BackgroundSubtractorGMG` es un algoritmo que combina una estimación estadística del fondo de la imagen junto con una segmentación Bayesiana píxel a píxel [29].
- `BackgroundSubtractorFGD` está disponible en la versión para CUDA. Utiliza la regla de decisión de Bayes para clasificar los elementos del fondo y los del primer plano atendiendo a sus vectores de características [19].

3.3. Posprocesado

Para mejorar los resultados de la extracción de fondo y preparar la imagen para la búsqueda de contornos, se han aplicado las siguientes técnicas:

Dilatación

Se trata de una operación morfológica por la cual se expanden las regiones luminosas de una imagen. Esto se consigue mediante la sustitución de cada

píxel por el más brillante de los vecinos considerados por el *kernel* (matriz utilizada para la convolución). De esta manera se consiguen unir las regiones de abejas que podían haberse roto [17].

Erosión

Se trata de la operación contraria a la anterior, expande las regiones oscuras de la imagen. Para ello se coge el valor mínimo de los valores considerados por el *kernel* [17].

La dilatación nos permite reconstruir las abejas, pero también aumenta su tamaño, aumentando el riesgo de solapamientos. Para evitar esto, se vuelve a reducir el tamaño de estas mediante una erosión.

En nuestro algoritmo aplicamos tres operaciones morfológicas seguidas:

1. **Erosión (3x3)**: elimina las piernas de las abejas.
2. **Dilatación (2x2)**: junta la cabeza de las abejas con su cuerpo que en numerosas ocasiones es separado durante la substracción de fondo.
3. **Erosión (3x3)**: recupera el tamaño inicial.

A continuación podemos ver el resultado de esta fase:

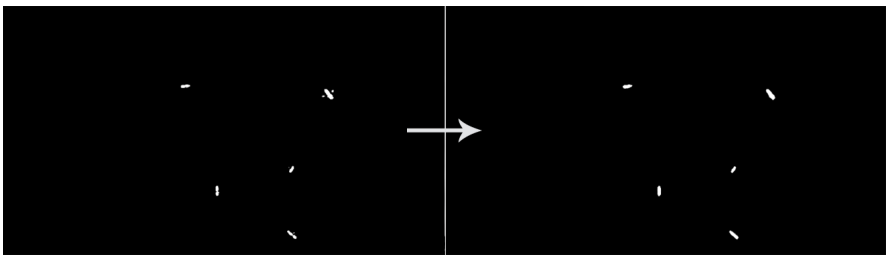

Figura 3.4: Resultado de la fase de posprocesado.

3.4. Detección y conteo de abejas

El último paso que realiza nuestro algoritmo de visión artificial es detectar cuáles de las regiones obtenidas en la fase anterior se correponden con abejas. Para ello, se realiza una búsqueda de contornos y se filtran por área.

Entendemos por contorno una línea curva que une todos los puntos continuos del borde de una región de un mismo color o intensidad.

La salida de la fase anterior es una imagen binaria con los objetos en movimiento en blanco y el fondo en negro. Por lo tanto, el objetivo de esta

fase es detectar todas las regiones blancas que puedan corresponderse con una abeja.

OpenCV provee la función `Imgproc.findContours()` para realizar la búsqueda de contornos. Esta toma una imagen binaria y devuelve una lista con todos los contornos encontrados. Para entender la función se necesita comprender una serie de conceptos: [31]

- **Jerarquía**: los contornos pueden ser independientes unos de otros, o poseer una relación padre-hijo cuando un contorno está dentro de otro. En la jerarquía se especifican las relaciones entre contornos.
- **Modo de obtención del contorno**: define cómo se van a obtener los contornos en cuestión de jerarquía [32].
  - `RETR_LIST`: devuelve todos los contornos en una lista, sin ninguna información de jerarquía entre ellos.
  - `RETR_EXTERNAL`: devuelve todos los contornos externos. Si algún contorno tiene contornos hijo, estos son ignorados.
  - `RETR_CCOMP`: devuelve los contornos agrupados en dos niveles de jerarquía. Un primer nivel en el que se encuentran todos los contornos exteriores. Y un segundo nivel con los contornos correspondientes a agujeros en los primeros.
  - `RETR_TREE`: devuelve todos los contornos creando un árbol completo con la jerarquía.
- **Método de aproximación de los contornos**: define el método que utiliza la función para almacenar los contornos [32].
  - `CHAIN_APPROX_NONE`: almacena todos los puntos del borde del contorno.
  - `CHAIN_APPROX_SIMPLE`: almacena sólo los puntos relevantes del contorno. Por ejemplo, si el contorno es una línea no se necesita almacenar todos los puntos de esta, con el punto inicial y el final basta. Esto es lo que realiza este método, eliminar todos los puntos redundantes y comprimirlos para que ocupe menos espacio.
  - `CV_CHAIN_APPROX_TC89_L1` y `CV_CHAIN_APPROX_TC89_KCOS`: aplican el algoritmo de aproximación de cadena de Teh-Chin, simplificando los polígonos que forman los contornos.
  - `CV_CHAIN_CODE`: almacena los contornos utilizando el código de cadenas de Freeman.

En nuestro caso, la configuración más adecuada es utilizar `RETR_EXTERNAL` y `CHAIN_APPROX_SIMPLE`. Ya que no nos interesa ningún contorno interno que pueda tener la abeja (y que en principio no debería tener) y tampoco nos es relevante el cómo se almacenan estos, sólo nos interesa el número.

Para evitar posibles falsos positivos, establecemos un umbral mínimo y máximo en el área del contorno. De esta manera, evitamos que contornos diminutos o grandes generados por ruidos o por objetos del entorno (moscas, pájaros, roedores...) sean contados cómo abejas.

En la siguiente imagen podemos ver la salida del algoritmo:

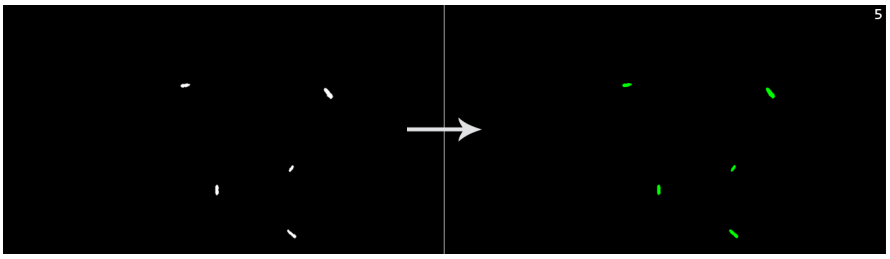

Figura 3.5: Resultado de la fase de detección y conteo de abejas.

En esta otra se puede apreciar como se descartan las tres moscas que hay en la imagen ya que su área es inferior al área mínima:

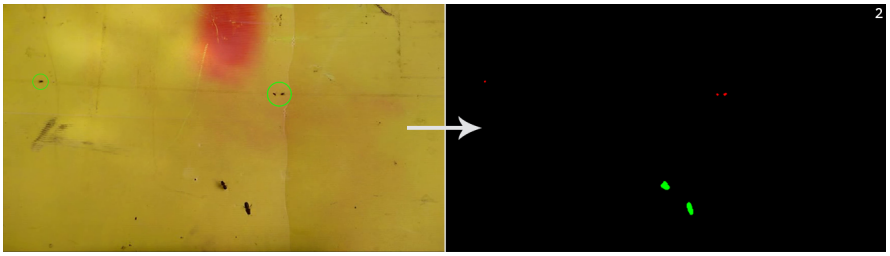

Figura 3.6: Resultado del algoritmo en una imágen con moscas.

Técnicas y herramientas

4.1. Metodologías

Scrum

Scrum es un marco de trabajo para el desarrollo de *software* que se engloba dentro de las metodologías ágiles. Aplica una estrategia de trabajo iterativa e incremental a través de iteraciones (*sprints*) y revisiones [39].

Test-Driven Development (TDD)

Es una práctica de desarrollo de *software* que se basa en la repetición de un ciclo corto de desarrollo: transformar requerimientos a test, desarrollar el código necesario para pasar los test y posteriormente refactorizar el código. Esta práctica obliga a los desarrolladores a analizar cuidadosamente las especificaciones antes de empezar a escribir código, fomenta la escritura de test, la simplicidad del código y aumenta la productividad. Como resultado se obtiene *software* más seguro y de mayor calidad [40].

Gitflow

Gitflow es un flujo de trabajo con Git que define un modelo estricto de ramas diseñado en torno a los lanzamientos de proyecto. En la rama *main* se hospeda la última versión estable del proyecto. La rama *develop* contiene los últimos desarrollos realizados para el siguiente lanzamiento. Por cada característica que se vaya a implementar se crea una *feature branch*. La preparación del siguiente lanzamiento se realiza en una *release branch*. Si aparece un fallo en producción, este se soluciona en una *hotfix branch* [9].

4.2. Patrones de diseño

Model-View-Presenter (MVP)

MVP es un patrón de arquitectura derivado del *Model–View–Controller* (MVC). Permite separar los datos internos del modelo de una vista pasiva y enlazarlos mediante el *presenter* que maneja toda la lógica de la aplicación [10]. Posee tres capas:

- ***Model***: almacena y proporciona los datos internos.
- ***View***: maneja la visualización de los datos (del modelo). Propaga las acciones de usuario al *presenter*.
- ***Presenter***: enlaza las dos capas anteriores. Sincroniza los datos mostrados en la vista con los almacenados en el modelo y actúa ante los eventos de usuario propagados por la vista.

Figura 4.7: Patrón MVP.

Patrón repositorio

El patrón repositorio proporciona una abstracción de la implementación del acceso a datos con el objetivo de que este sea transparente a la lógica de negocio de la aplicación. Por ejemplo, las fuentes de datos pueden ser una base de datos, un *web service*, etc. El repositorio media entre la capa de acceso a datos y la lógica de negocio de tal forma que no existe ninguna dependencia entre ellas. Consiguiendo desacoplar, mantener y testear más fácilmente el código y permitiendo la reutilización del acceso a datos desde cualquier cliente [25].

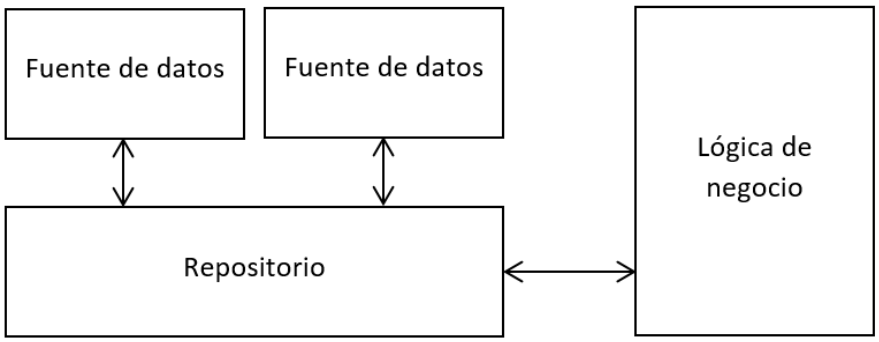

Figura 4.8: Patrón Repositorio.

4.3. Control de versiones

- Herramientas consideradas: Git y Subversion.
- Herramienta elegida: Git.

Git es un sistema de control de versiones distribuido. A día de hoy, es el sistema con mayor número de usuarios. La principal diferencia con Subversion es su carácter descentralizado, que permite a cada desarrollador tener una copia en local del repositorio completo. Git está licenciado bajo la licencia de software libre GNU LGPL v2.1.

4.4. *Hosting* del repositorio

- Herramientas consideradas: GitHub, Bitbucket y GitLab.
- Herramienta elegida: GitHub.

GitHub es la plataforma web de hospedaje de repositorios por excelencia. Ofrece todas las funcionalidades de Git, revisión de código, documentación, *bug tracking*, gestión de tareas, *wikis*, red social... y numerosas integraciones con otros servicios. Es gratuita para proyectos *open source*.

Utilizamos GitHub como plataforma principal donde hospedamos el código del proyecto, la gestión de proyecto (gracias a ZenHub) y la documentación. Además, el repositorio está integrado con varios servicios de integración continua.

4.5. Gestión del proyecto

- Herramientas consideradas: ZenHub, Trello, Waffle, VersionOne, XP-Dev y GitHub Projects.

- Herramienta elegida: ZenHub.

ZenHub es una herramienta de gestión de proyectos totalmente integrada en GitHub. Proporciona un tablero canvas en donde cada tarea representada se corresponde con un *issue* nativo de GitHub. Cada tarea se puede priorizar dependiendo de su posición en la lista, se le puede asignar una estimación, uno o varios responsables y el *sprint* al que pertenece. ZenHub también permite visualizar el gráfico *burndown* de cada *sprint*. Es gratuita para proyectos pequeños (máx. 5 colaboradores) o proyectos *open source*.

4.6. Comunicación

- Herramientas consideradas: email y Slack.
- Herramienta elegida: Slack.

Slack es una herramienta de colaboración de equipos que ofrece salas de chat, mensajes directos y llamadas VoIP. Posee un buscador que permite encontrar todo el contenido generado dentro de Slack. Además, ofrece un gran número de integraciones con otros servicios. En nuestro proyecto vamos a utilizar la integración con GitHub para crear un canal que sirva de *log* de todas las acciones realizadas en GitHub. Slack ofrece una versión gratuita que provee las características principales.

4.7. Entorno de desarrollo integrado (IDE)

Java

- Herramientas consideradas: IntelliJ IDEA y Eclipse.
- Herramienta elegida: IntelliJ IDEA.

IntelliJ IDEA es un IDE para Java desarrollado por JetBrains. Posee un gran número de herramientas para facilitar el proceso de escritura, revisión y refactorización del código. Además, permite la integración de diferentes herramientas y posee un ecosistema de *plugins* para ampliar su funcionalidad. Su versión *community* está disponible bajo la licencia Apache 2. Aunque también es posible adquirir la versión *Ultimate* gratuitamente si se es estudiante.

Android

- Herramientas consideradas: Android Studio y Eclipse.
- Herramienta elegida: Android Studio.

Android Studio es el IDE oficial para el desarrollo de aplicaciones Android. Está basado en IntelliJ IDEA de JetBrains. Proporciona soporte para Gradle, emulador, editor de *layouts*, refactorizaciones específicas de Android, herramientas Lint para detectar problemas de rendimiento, uso, compatibilidad de versión, etc. Se distribuye bajo la licencia Apache 2.

Markdown

- Herramientas consideradas: StackEdit y Haroopad.
- Herramienta elegida: Haroopad.

Haroopad es un editor de documentos Markdown. Soporta Github Flavored Markdown y Mathematics Expression, además de contar con un gran número de extensiones. Se distribuye bajo licencia GNU GPL v3.0.

LaTeX

- Herramientas consideradas: ShareLaTeX y Texmaker.
- Herramienta elegida: Texmaker.

Texmaker es un editor gratuito y multiplataforma para LaTeX. Integra la mayoría de herramientas necesarias para la escritura de documentos en LaTeX(PdfLaTeX , BibTeX, makeindex, etx). Además, incluye corrector ortográfico, auto-completado, resaltado de sintaxis, visor de PDFs integrado, etc. Está licenciado bajo GNU GPL v2.

4.8. Documentación

- Herramientas consideradas: LaTeX, Markdown, reStructuredText y Microsoft Word
- Herramienta elegida: Markdown + reStructuredText + LaTeX.

La documentación se ha desarrollado en Markdown y reStructuredText para integrarla con el servicio de documentación continua Read the Docs. Una vez terminada, se ha exportado a LaTeXutilizando el conversor Pandoc.

Markdown es un lenguaje de marcado ligero en texto plano que puede ser exportado a numerosos formatos como HTML o PDF. Su filosofía es que el lenguaje de marcado sea fácil de escribir y leer. Markdown es ampliamente utilizado para la escritura de archivos README, en foros como StackOverflow o en herramientas de comunicación como Slack.

reStructuredText es uno de los leguajes de marcado ligero en los que se inspiró Markdown. Su principal aplicación es la escritura de documentación de Python junto con el sistema de generación de documentación Sphinx.

LaTeXes un sistema de composición de textos que genera documentos con una alta calidad tipográfica. Es ampliamente utilizado para la generación de artículos y libros científicos, principalmente por su potencia a la hora de representar expresiones matemáticas.

4.9. Servicios de integración continua

Compilación y testeo

- Herramientas consideradas: TravisCI y CircleCI.
- Herramienta elegida: TravisCI.

Travis es una plataforma de integración continua en la nube para proyectos alojados en GitHub. Permite realizar una *build* del proyecto y testearla automáticamente cada vez que se realiza un *commit*, devolviendo un informe con los resultados. Es gratuita para proyectos *open source*.

Cobertura de código

- Herramientas consideradas: Coveralls y Codecov.
- Herramienta elegida: Codecov.

Codecov es una herramienta que permite medir el porcentaje de código que está cubierto por un test. Además, realiza representaciones visuales de la cobertura y gráficos de su evolución. Posee una extensión de navegador para GitHub que permite visualizar por cada archivo de código que líneas están cubiertas por un test y cuáles no. Es gratuita para proyectos *open source*.

Calidad del código

- Herramientas consideradas: Codeclimate, SonarQube y Codacy.
- Herramientas elegidas: Codeclimate y SonarQube.

Codeclimate es una herramienta que realiza revisiones de código automáticamente. Es gratuita para proyectos *open source*. En nuestro proyecto hemos activado los siguientes motores de chequeo: checkstyle, fixme, markdownlint y pmd.

SonarQube es una plataforma de código abierto para la revisión continua de la calidad de código. Permite detectar código duplicado, violaciones de estándares, cobertura de tests unitarios, *bugs* potenciales, etc.

Revisión de dependencias

VersionEye es una herramienta que monitoriza las dependencias del proyecto y envía notificaciones cuando alguna de estas está desactualizada, es vulnerable o viola la licencia del proyecto. Posee una versión gratuita con ciertas limitaciones.

Documentación continua

Read the Docs es un servicio de documentación continua que permite crear y hospedar una página web generada a partir de los distintos ficheros Markdown o reStructuredText de la documentación. Cada vez que se realiza un *commit* en el repositorio se actualiza la versión hospedada. La página web posee un buscador, da soporte para diferentes versiones del proyecto y soporta internacionalización. Además, permite exportar la documentación en varios formatos (pdf, epub, html, etc.). El servicio es totalmente gratuito, sostenido por donaciones y subscripciones *Gold*.

4.10. Sistemas de construcción automática del software

Maven

Maven es una herramienta para automatizar el proceso de construcción del *software* (compilación, testeo, empaquetado, etc.) enfocada a proyectos Java. Básicamente describe cómo se tiene que construir el *software* y cuáles son sus dependencias.

Gradle

Gradle es una herramienta similar a Maven pero basada en el lenguaje de programación orientado a objetos Groovy. El sistema de construcción de Android Studio está basado en Gradle y es actualmente el único soportado de forma oficial para Android.

4.11. Librerías

Android Support Library

La librería de soporte de Android facilita algunas características que no se incluyen en el *framework* oficial. Proporciona compatibilidad a versiones antiguas con las últimas características, incluye elementos para la interfaz adicionales y utilidades extra.

Espresso

Espresso es un framework de *testing* para Android incluido en la librería de soporte para *testing* en Android. Provee una API para escribir UI test que simulen las interacciones de usuario con la app.

Google Guava

Google Guava agrupa un conjunto de librerías comunes para Java. Proporciona utilidades básicas para tareas cotidianas, una extensión del *Java collections framework* (JCF) y otras extensiones como programación funcional, almacenamiento en caché, objetos de rango o *hashing*.

Google Play Services

Google Play Services es una librería que permite a las aplicaciones de terceros utilizar características de aplicaciones de Google como Maps, Google+, etc. En nuestro caso se ha hecho uso de su servicio de localización, que utiliza varias fuentes de datos (GPS, red y *wifi*) para ubicar el dispositivo rápidamente.

JavaFX

JavaFX es una librería para la creación de interfaces gráficas en Java.

JUnit

JUnit es un *framework* para Java utilizado para realizar pruebas unitarias.

Material Design

Material Design es una guía de estilos enfocada a la plataforma Android, pero aplicable a cualquier otra plataforma. Fue presentada en el Google I/O 2014 y se adoptó en Android a partir de la versión 5.0 (Lollipop). Se basa en objetos materiales, piezas colocadas en un espacio (lugar) y con un tiempo (movimiento) determinado.

Mockito

Mockito es un *framework* de *mocking* que permite crear objetos *mock* fácilmente. Estos objetos simulan parte del comportamiento de una clase. Mockito está basado en EasyMock, mejorando su sintaxis haciendo los test más simples y fáciles de leer y con mensajes de error descriptivos.

MPAndroidChart

MPAndroidChart es una librería para la creación de gráficos en Android.

OpenCV

OpenCV es un paquete *Open Source* de visión artificial que contiene más de 2500 librerías de procesamiento de imágenes y visión artificial, escritas en C/C++ a bajo/medio nivel. Se distribuye gratuitamente bajo una licencia *BSD* desde hace más de una década. Posee una comunidad de más de 50.000 usuarios alrededor de todo el mundo y se ha descargado más de 8 millones de veces.

Aunque OpenCV está escrito en C/C++ posee *wrappers* para varias plataformas, entre ellas Android, en donde da soporte a las principales arquitecturas de CPU. Desde hace unos años, también soporta CUDA para el desarrollo en GPU tanto en escritorio como en móvil, aunque en esta última el soporte es todavía reducido.

OpenWeatherMaps

OpenWeatherMap es un servicio online que proporciona información meteorológica. Está inspirado en OpenStreetMap y su filosofía de hacer accesible la información a la gente de forma gratuita. Utiliza distintas fuentes de datos desde estaciones meteorológicas oficiales, de aeropuertos, radares e incentiva a los propietarios de estaciones meteorológicas a conectarlas a su red. Proporciona una API que permite realizar hasta 60 llamadas por segundo de forma gratuita.

PowerMock

PowerMock es una librería de *testing* que permite la creación de *mocks* de métodos estáticos, constructores, clases finales o métodos privados.

Realm

Realm es una base de datos orientada a objetos enfocada a dispositivos móviles. Se definen como la alternativa a SQLite y presumen de ser más rápidos que cualquier ORM e incluso que SQLite puro. Posee una API muy intuitiva que facilita en gran medida el acceso a datos.

4.12. Página web

GitHub Pages

GitHub Pages es un servicio de hosting estático que permite a proyectos

que utilicen un repositorio de GitHub hospedar su página web en el propio repositorio. Permite utilizar Jekyll, un generador de sitios estáticos. No soporta tecnologías del lado de servidor como PHP, Ruby, Python, etc.

Bootstrap

Bootstrap es un *framework* para desarrollo *front-end*. Contiene una serie de componentes ya implementados que facilitan y agilizan el diseño. Está desarrollado siguiendo la filosofía *mobile first*.

4.13. Otras herramientas

Mendeley

Mendeley es un gestor de referencias bibliográficas. Permite añadir referencias de varias formas, visualizar los documentos, etiquetarlos, compartirlos, etc. Posteriormente se puede exportar todo el catálogo a un fichero BibTex para ser utilizadas desde LaTeX.

Creately

Creately es una aplicación web que permite crear todo tipo de diagramas altamente personalizables. Aunque posee una versión gratuita limitada, se optó por pagar un mes de subscripción al valorar que realmente iba a ser de utilidad.

Aspectos relevantes del desarrollo del proyecto

En este apartado se van a recoger los aspectos más importantes del desarrollo del proyecto. Desde las decisiones que se tomaron y sus implicaciones, hasta los numerosos problemas a los que hubo que enfrentarse y cómo se solucionaron.

5.1. Inicio del proyecto

La idea del proyecto surgió del afán de encontrar un tema con el que poder aunar mi formación técnica y mis aficiones.

Mi padre me transmitió el interés por la apicultura desde bien pequeño y, posteriormente, llegué a trabajar en una empresa de apicultura profesional. Esto me hizo ser consciente de los problemas con los que lidia día a día un apicultor. Por otro lado, los conocimientos adquiridos durante mis estudios de Ingeniería Informática, me posibilitaron idear soluciones tecnológicas a alguno de estos problemas.

Tras formalizar la idea del proyecto y recibir el visto bueno de los tutores, nos pusimos manos a la obra.

Figura 5.9: Logo GoBees.

5.2. Metodologías

Desde el primer momento se dedicaron grandes esfuerzos para que la realización del proyecto se llevase a cabo de la manera más profesional posible. Para ello, se siguieron varias metodologías y procesos, expuestos a continuación.

Para la gestión del proyecto se utilizó una metodología ágil, en concreto Scrum. Aunque no se siguió al 100 % al tratarse de un proyecto educativo (no éramos un equipo de 4 a 8 personas, no hubo reuniones diarias, etc.), sí que se aplicó una filosofía ágil en líneas generales:

- Se siguió una estrategia de desarrollo incremental a través de iteraciones (*sprints*) y revisiones.
- La duración media de los *sprints* fue de una semana.
- Al finalizar cada *sprint* se entregaba una parte del producto operativo (incremento).
- Se realizaban reuniones de revisión al finalizar cada *sprint* y al mismo tiempo de planificación del nuevo *sprint*.
- En la planificación del *sprint* se generaba una pila de tareas a realizar.
- Estas tareas se estimaban y priorizaban en un tablero *canvas* (ZenHub).
- Para monitorizar el progreso del proyecto se utilizó gráficos *burndown*.

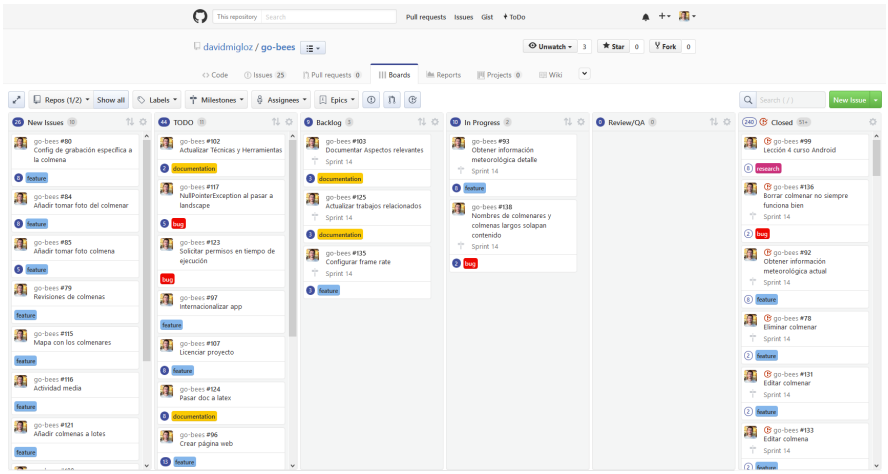

Figura 5.10: Tablero *canvas*.

Para el diseño del algoritmo de visión artificial se utilizó una metodología de ensayo y error. Se barajaban diferentes alternativas y se iban verificando empíricamente su eficacia y eficiencia.

Para el desarrollo de la aplicación Android se intentó utilizar *Test-Driven Development* (TDD) para fomentar la escritura de test y mejorar la calidad del *software*. La mayoría de los módulos de la aplicación se implementaron exitosamente siguiendo esta metodología. Sin embargo, en las partes más complejas de esta, como el servicio de monitorización, se hacía muy engorroso la escritura de test (ya complicada de por sí en Android), por lo que finalmente se omitieron al penalizar notablemente la productividad.

5.3. Formación

El proyecto requería una serie de conocimientos técnicos de los que no se disponía en un principio. Sobre todo, relacionados con visión artificial, OpenCV y Android. A continuación, se enumeran los principales recursos didácticos que se utilizaron.

Para la formación en visión artificial y OpenCV se leyeron los siguientes libros:

- *Android Application Programming with OpenCV 3* (Joseph Howse) [14].
- *Mastering OpenCV Android Application Programming* (Salil Kapur y Nisarg Thakkar) [17].
- *Learning Image Processing with OpenCV* (Ismael Serrano Gracia, Jesús Salido Tercero y José Luis Espinosa Aranda) [12].
- *OpenCV 3.0 Computer Vision with Java* (Daniel Lélis Baggio) [1].

Para la formación en Android se realizaron los siguientes cursos online:

- *Android Development for Beginners: How to Make Apps* (Udacity) [18].
- *Developing Android Apps* (Udacity) [11].
- *Android Testing Codelab* (Google) [15].

También cabe destacar la importancia que tuvo la comunidad StackOverflow para la resolución de los diferentes problemas que surgieron durante el desarrollo.

5.4. Desarrollo del algoritmo

Una gran parte de los recursos del proyecto se dedicaron al desarrollo del algoritmo de visión artificial para la monitorización de la actividad de vuelo de una colmena.

El problema a resolver poseía una serie de condicionantes que dificultaban el análisis:

- Cada abeja ocupa una porción muy pequeña de la imagen.
- Las condiciones lumínicas varían a lo largo del día o de la época del año.
- Existen sombras producidas por la cámara o por las propias abejas.
- A 20 fps una abeja volando puede recorrer una distancia significativa entre fotogramas.
- Un grupo de abejas puede estar confinado, dificultando su segmentación.

El desarrollo comenzó con una búsqueda bibliográfica sobre el tema. En esta, se encontraron varios artículos relacionados que nos dieron una idea de las técnicas que podíamos probar. También se tuvo una reunión con un experto en visión artificial que nos proporcionó su punto de vista sobre cómo abordar el problema.

El primer punto a abordar fue la toma de las imágenes. Se decidió que la toma se debía de hacer con un trípode en posición cenital a la colmena. De esta manera, se disminuía las diferencias de tamaño por perspectiva, no se obstaculizaba el vuelo de las abejas y se facilitaba el análisis. Además, era aconsejable cubrir el suelo con alguna superficie uniforme para mejorar la segmentación de las abejas.

Figura 5.11: Colocación del *smartphone* en la colmena.

En un primer momento, el desarrollo del algoritmo se iba a realizar directamente sobre la plataforma Android. Sin embargo, rápidamente nos dimos cuenta que no era lo más adecuado si queríamos seguir una metodología basada en el ensayo y error, debido a los elevados tiempos de compilación y a la poca flexibilidad de la plataforma.

Finalmente se decidió desarrollar el algoritmo directamente en Java, ya que nos proporcionaba una mayor flexibilidad y, además, migrar el código a Android posteriormente sería una tarea bastante trivial.

Para facilitar el desarrollo del algoritmo y del testeo de las diferentes alternativas se realizó una aplicación Java que permitía parametrizar en tiempo real las diferentes etapas del algoritmo.

Además de permitir variar la parametrización, la aplicación nos permitía observar la salida de cada etapa del algoritmo y su tiempo de computación.

Es importante recalcar que los tiempos de computación obtenidos en la plataforma había que multiplicarlos por cuatro si se quería obtener una estimación de lo que equivaldría ejecutarlo en el dispositivo móvil. Por lo que los márgenes con los que se jugaba no eran demasiado grandes.

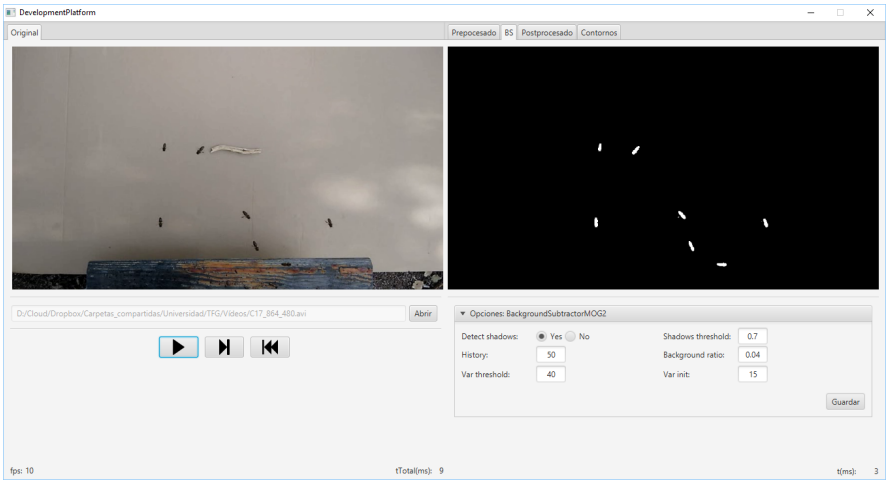

Figura 5.12: Plataforma de desarrollo del algoritmo.

Tras varias iteraciones, se consiguió obtener un algoritmo bastante robusto que poseía las siguientes etapas:

1. **Preprocesado:** conversión de RGB a escala de grises y desenfoque Gaussiano para facilitar el procesado.
2. **Substracción del fondo:** extracción de los elementos en movimiento utilizando el algoritmo `BackgroundSubtractorMOG2`.
3. **Posprocesado:** mejora de la salida de la etapa anterior mediante varias iteraciones de erosión y dilatación.
4. **Detección y conteo de abejas:** localización de los contornos pertenecientes a abejas en base al área y conteo de los mismos.

5.5. Desarrollo de la *app*

El desarrollo de la aplicación Android se realizó de forma incremental, publicando una *release* al finalizar cada *sprint*.

La primera tarea consistió en migrar el algoritmo de visión artificial a la plataforma Android. A primera vista, no parecía una tarea complicada. Sin embargo, nos encontramos con varios *bugs* que, unidos a la mala documentación de OpenCV para Android, dificultó considerablemente la labor.

El primer *bug* (*issue* #26) consistía en que al llamar a algunas funciones del núcleo de OpenCV, se producía un error con un mensaje que hacía referencia a una señal que variaba entre ejecuciones, lo cual hizo muy difícil encontrar el origen. Finalmente, se dio con la fuente del problema, el cual tenía su origen en que la versión de la aplicación OpenCV Manager distribuida en el Google Play contaba con una versión corrupta de la librería OpenCV 3.1.

Se notificó al equipo de OpenCV a través de su gestor de incidencias y finalmente solucionaron el problema en la nueva versión OpenCV 3.2.

El segundo *bug* (*issue* #27) hacía que la aplicación fallase cuando se compilaba para dispositivos con una versión de Android inferior a Lollipop, debido a la API de la cámara. Tras publicar el caso en StackOverflow, un usuario sugirió que podía estar relacionado con el *Instant Run* de Android Studio. Y así fue; desactivando esta característica, la aplicación no fallaba. Se describió el caso en el gestor de incidencias de Android y, a día de hoy (enero 2017), el *bug* se encuentra resuelto y está a la espera de ser incorporado en futuras *releases* del *framework*.

Una vez solventados ambos *bugs,* se logró que el algoritmo funcionase sin problemas en la nueva plataforma.

Cabe destacar que como OpenCV no distribuía oficialmente la librería a través de ningún repositorio que permitiese utilizarla directamente como dependencia Gradle, se creó uno propio [21].

Para el diseño de la arquitectura de la *app* se siguió el patrón de arquitectura *Model-View-Presenter* (MVP), que permite separar los datos internos del modelo de una vista pasiva y agrupar toda la lógica de la aplicación en una capa intermedia llamada *presenter*. De esta manera se consigue un código muy desacoplado, haciendo que este sea más fácil de testear y mantener.

En cuanto a la persistencia de datos, se optó por utilizar Realm. Se trata de una base de datos orientada a objetos que proporciona una API para trabajar directamente con la capa de persistencia. Es multiplataforma y presume de ser más rápida que SQLite.

Para la obtención de la información meteorológica se escogió la API proporcionada por *OpenWeatherMaps*, la cual nos permitía realizar hasta 60 llamadas por minuto de forma gratuita.

El acceso a datos se centralizó utilizando el patrón repositorio, que abstrae la lógica de negocio de la fuente de datos. Todo el acceso a datos se centraliza en el repositorio y es este quien decide de qué fuente los obtiene (base de datos, internet, etc.). Además, se incorporó una capa de caché en este punto con el fin de agilizar la navegación por la *app*.

El siguiente esquema resume la arquitectura de la aplicación:

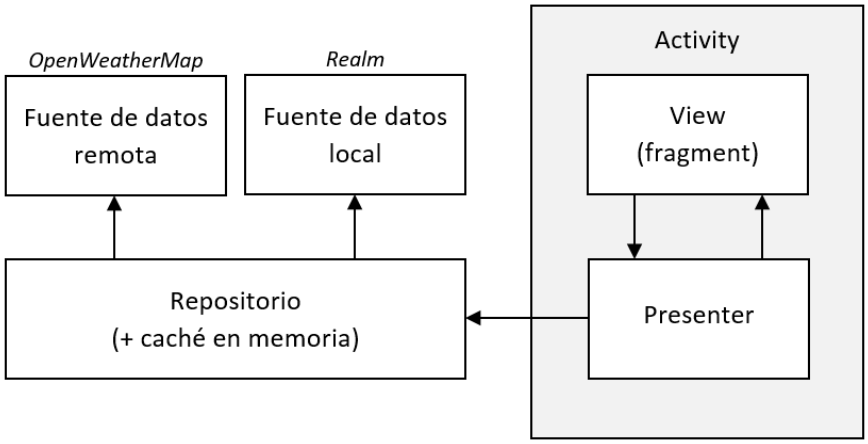

Figura 5.13: Arquitectura de la aplicación.

Surgieron problemas al implementar el algoritmo de monitorización como un servicio de Android para que el usuario pudiese apagar la pantalla durante la monitorización. El acceso a la cámara proporcionado por OpenCV era en sí una vista, y las vistas no pueden ser utilizadas en servicios. Finalmente se optó por realizar una implementación propia que accediese directamente a la API de la cámara y convirtiese los fotogramas al formato de OpenCV.

En cuanto al diseño de la aplicación, se dedicaron grandes esfuerzos a la usabilidad y accesibilidad de la misma. Se siguieron las directrices de diseño recogidas en la guía de Material Design en cuanto a estilos, disposición de los elementos, tipos de componentes, patrones de navegación, gestos, etc.

Por último, se internacionalizó la aplicación a los siguientes idiomas: español, inglés, catalán, polaco y árabe. Para ello se utilizó la herramienta Toolkit del Traductor de Google que permite realizar una primera traducción automática de los diferentes textos de la aplicación y posteriormente una revisión colaborativa de los resultados de esta. Para la revisión se recurrió a amistades nativas en los diferentes idiomas.

5.6. *Testing*

En lo relativo al *testing*, podemos diferenciar el testeo del algoritmo del testeo de la *app*.

Para testear el error cometido por el algoritmo era necesario contar con fragmentos de vídeo etiquetados con el número de abejas presentes en cada fotograma. Como esta labor era muy tediosa, se desarrolló una aplicación Java para agilizar el etiquetado de los fotogramas.

La aplicación iba mostrando los diferentes fotogramas al usuario y este indicaba con el ratón los píxeles pertenecientes a abejas. Finalmente permitía exportar los datos en un archivo CSV.

Figura 5.14: Plataforma de conteo de abejas.

Se etiquetaron fragmentos de vídeo correspondientes a tres situaciones distintas:

- **Caso 1** (vídeo #C14): actividad media, sombras de abejas y moscas.
- **Caso 2** (vídeo #C17): actividad media, cambios de iluminación y sombras de árboles.
- **Caso 3** (vídeo #C5): alta actividad, solapamiento, sombras y fondo no uniforme.

Tras ejecutar el algoritmo se obtuvieron los siguientes errores relativos (considerando como error absoluto la distancia entre la medida esperada y la obtenida):

- **Caso 1:** 2,43 %.
- **Caso 2**: 0,89 %.
- **Caso 3**: 4,48 %.

Se puede observar que en la situación menos favorable el error es menor a un 5 %, precisión más que suficiente para la finalidad de los datos.

En cuanto al tiempo medio de ejecución del algoritmo para cada fotograma era de 25 ms en el equipo de pruebas (Intel i7-3610QM) y de 100 ms cuando se ejecutaba en un *smartphone* (Xiaomi Mi4).

Por otra parte, la aplicación se testeó mediante test unitarios, test de integración y test de interfaz. La mayor parte de los test unitatios se realizaron contra los *presenters*, que son los que poseen la lógica de negocio de la *app*. Los test de interfáz se ejecutaron en siete dispositivos diferentes, uno por cada versión de Android que soporta la *app* (API 19-25).

Además, se configuraron una serie de servicios de integración continua, de tal forma, que cada vez que se realizaba un *commit* en el repositorio, se ejecutaban las siguientes tareas:

1. **Travis**: realizaba una compilación del proyecto, ejecutaba los test unitarios, ejecutaba *Lint*, ponía en marcha un emulador de Android, y ejecutaba los Android test sobre este. Al finalizar, enviaba los resultados a Codecov y SonarQube.
2. **Codecov**: realizaba un análisis sobre la cobertura de los test unitarios.
3. **CodeClimate**: ejecutaba cuatro motores de chequeo (*checkstyle*, *fixme*, *pmd* y *markdownlint*) sobre el código para detectar posibles problemas o vulnerabilidades en él.
4. **SonarQube**: analizaba código duplicado, violaciones de estándares, cobertura de tests unitarios, bugs potenciales, etc.
5. **VersionEye**: chequeaba todas las dependencias utilizadas en la aplicación y comprobaba si estaban actualizadas, si tenían algún problema de seguridad conocido, o si violaban la licencia del proyecto.

Figura 5.15: Servicios de integración continua.

Por último, cabe comentar algunas estadísticas del proyecto:

Concepto	Valor
Total de líneas	TODO
Líneas de código	TODO
Comentarios	TODO
Líneas en blanco	TODO
Número de clases	TODO
Número de XML	TODO
Cobertura total test unitarios	TODO
Cobertura test unitarios algoritmo	TODO

Tabla 5.1: Estadísticas del proyecto.

5.7. Documentación

En un primer momento, se decidió escribir la documentación en formato Markdown, utilizando las *wikis* que proporciona GitHub en el repositorio. De esta forma, la documentación podía ser visualizada directamente desde GitHub, sin necesidad de tener que compilarla con cada modificación.

Posteriormente, se optó por implementar un sistema de documentación continua integrado en el repositorio, en concreto ReadTheDocs. De tal forma, que la documentación se escribía en archivos Markdown dentro del repositorio y este servicio generaba una página web (go-bees.readthedocs.io) que se actualizaba cada vez que se realizaba un *commit*.

No obstante, los tutores preferían obtener la documentación en formato PDF para su corrección, por lo que finalmente se optó por utilizar Sphinx junto con ReadTheDocs.

Sphinx es un generador de documentación que permite exportar la documentación en varios formatos, entre ellos HTML y PDF. Desafortunadamente, no soportaba Markdown como formato de entrada, por lo que hubo que migrar la documentación al formato reStructuredText. Esta conversión se realizó con la herramienta Pandoc.

Con todo configurado, ahora ReadTheDocs generaba automáticamente la página web y un PDF actualizado con los últimos cambios realizados en la documentación.

Figura 5.16: Página web de documentación generada con ReadTheDocs.

Para la exportación final de la memoria se utilizó el conversor Pandoc, con objeto de transformar la documentación del formato reStructuredText a LaTeX. Algunos elementos, como las citas o las imágenes, no eran convertidos correctamente, por lo que se tuvo que hacer uso de expresiones regulares.

La totalidad de este tedioso proceso se realizó bajo la idea de que cualquier proyecto comercial tiene su documentación accesible desde una página web, y que además necesita estar acorde con la versión del proyecto. Sin embargo, para este caso concreto en el que el entregable final es un PDF con un formato determinado, el montar todo este sistema ha supuesto una sobrecarga notable.

5.8. Publicación

En cuanto la aplicación estuvo lista para pasar a producción, se publicó en Google Play.

TODO meter captura Google Play.

Además, se desarrolló una página web promocional (gobees.io), donde se describen las características de la aplicación, los manuales de usuario, y el enlace de descarga, entre otras cosas.

TODO meter captura página web.

Por último, se crearon perfiles en las principales redes sociales para promocionar la aplicación.

5.9. Reconocimientos

Durante el desarrollo del proyecto se obtuvieron varios reconocimientos:

- **Beca de colaboración con departamentos**: se me concedió esta beca para profundizar en el algoritmo desarrollado y publicar un artículo científico sobre él.
- **Prototipos Orientados al Mercado**: GoBees resultó ganador de uno de los tres premios de la convocatoria de Prototipos Orientados al Mercado realizada por el Vicerrectorado de Investigación y Transferencia del Conocimiento de la Universidad de Burgos.
- **YUZZ**: GoBees fue elegido para participar en el programa YUZZ 2017, patrocinado por el Banco Santander para el impulso del talento joven y el espíritu emprendedor.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] Daniel Lélis Baggio. *OpenCV 3.0 Computer Vision with Java*. Packt Publishing, July 2015.

[2] Jason Campbell, Lily Mummert, and Rahul Sukthankar. Video Monitoring of Honey Bee Colonies at the Hive Entrance. *Visual observation analysis of animal insect behavior ICPR*, 8:1–4, 2008. URL http://homepages.inf.ed.ac.uk/rbf/VAIB08PAPERS/vaib9{_}mummert.pdf.

[3] Jennifer M Campbell, Douglas C Dahn, and Daniel A J Ryan. Capacitance-based sensor for monitoring bees passing through a tunnel. *Measurement Science and Technology*, 16(12):2503–2510, 2005. ISSN 0957-0233. doi: 10.1088/0957-0233/16/12/015. URL http://stacks.iop.org/0957-0233/16/i=12/a=015https://docs.google.com/file/d/0Bz7WbjRT9mlmTW1ac3R0azEycDA/view.

[4] Guillaume Chiron, Petra Gomez-Krämer, and Michel Ménard. Detecting and tracking honeybees in 3D at the beehive entrance using stereo vision. *EURASIP Journal on Image and Video Processing*, 2013(1):59, 2013. ISSN 1687-5281. doi: 10.1186/1687-5281-2013-59. URL http://jivp.eurasipjournals.com/content/2013/1/59https://hal.inria.fr/hal-00923374/document.

[5] Guillaume Chiron, Petra Gomez-Krämer, Michel Ménard, and Fabrice Requier. 3D tracking of honeybees enhanced by environmental context. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8156 LNCS, pages 702–711. Springer Berlin Heidelberg, 2013. ISBN 9783642411809. doi: 10.1007/978-3-642-41181-6_71. URL http://link.springer.com/10.1007/978-3-642-41181-6{_}71.

[6] Express Label Company. Barcode labels many uses, 2009. URL http://www.uprintlabels.com/barcode-labels-uses.html. [Online; Accedido 02-Noviembre-2016].

[7] Axel Decourtye, James Devillers, Pierrick Aupinel, François Brun, Camille Bagnis, Julie Fourrier, and Monique Gauthier. Honeybee tracking with microchips: A new methodology to measure the effects of pesticides. *Ecotoxicology*, 20(2):429–437, 2011. ISSN 09639292. doi: 10.1007/s10646-011-0594-4.

[8] Luis del Valle Hernández. Detección de movimiento con opencv y python, 2016. URL http://programarfacil.com/blog/vision-artificial/deteccion-de-movimiento-con-opencv-python/. [Online; Accedido 5-Octubre-2016].

[9] Vincent Driessen. Gitflow - a successful git branching model, 2010. URL http://nvie.com/posts/a-successful-git-branching-model/. [Online; Accedido 26-Enero-2017].

[10] Martin Fowler. Gui architectures - mvc and mvp, 2006. URL https://martinfowler.com/eaaDev/uiArchs.html. [Online; Accedido 22-Enero-2017].

[11] Dan Galpin, Lyla Fujiwara, Reto Meier, Asser Samak, James Williams, Cezanne Camacho, and Michael Lustig. Developing Android Apps | Udacity, 2016. URL https://www.udacity.com/course/new-android-fundamentals--ud851. [Online; Accedido 5-Octubre-2016].

[12] Gloria Bueno Garcia, Oscar Deniz Suarez, Jose Luis Espinosa Aranda, Jesus Salido Tercero, and Ismael Serrano Gracia. *Learning Image Processing with OpenCV*. Packt Publishing - ebooks Account, March 2015. ISBN 978-1-78328-765-9.

[13] Greenpeace Laboratories Research. Bees in Decline put pollinators and agriculture. page 48, 2013. URL http://www.greenpeace.org/switzerland/Global/international/publications/agriculture/2013/BeesInDecline.pdf.

[14] Joseph Howse. *Android Application Programming with OpenCV 3*. Packt Publishing, June 2015. ISBN 978-1-78528-538-7.

[15] Google Inc. Android Testing Codelab, 2016. URL https://codelabs.developers.google.com/codelabs/android-testing/. [Online; Accedido 5-Octubre-2016].

[16] J. K. Kaplan. Colony Collapse Disorder - A Complex Buzz. *Agricultural Research*, (June):8–11, 2008. ISSN 00027626. URL https://agresearchmag.ars.usda.gov/AR/archive/2008/May/colony0508.pdf.

[17] Salil Kapur. *Mastering OpenCV Android Application Programming*. Packt Publishing, July 2015.

[18] Katherine Kuan, Kunal Chawla, and Lyla Fujiwara. Android Development for Beginners: How to Make Apps | Udacity, 2016. URL https://www.udacity.com/course/android-development-for-beginners--ud837. [Online; Accedido 5-Octubre-2016].

[19] Liyuan Li, Weimin Huang, Irene Y. H. Gu, and Qi Tian. Foreground Object Detection from Videos Containing Complex Background. In *Proceedings of the Eleventh ACM International Conference on Multimedia*, MULTIMEDIA '03, pages 2–10, New York, NY, USA, 2003. ACM. ISBN 978-1-58113-722-4. doi: 10.1145/957013.957017. URL http://doi.acm.org/10.1145/957013.957017.

[20] David Miguel Lozano. Gobees en google play, 2016. URL https://play.google.com/store/apps/details?id=com.davidmiguel.gobees. [Online; Accedido 25-Enero-2017].

[21] David Miguel Lozano. Repositorio de las herramientas desarrolladas para gobees en github, 2016. URL https://github.com/davidmigloz/go-bees-prototypes. [Online; Accedido 25-Enero-2017].

[22] David Miguel Lozano. Repositorio de gobees en github, 2016. URL https://github.com/davidmigloz/go-bees/. [Online; Accedido 25-Enero-2017].

[23] David Miguel Lozano. Sitio web de gobees, 2016. URL http://gobees.io/. [Online; Accedido 25-Enero-2017].

[24] A. E. Lundie. The flight activities of the honeybee. *United States Department of Agriculture*, 1328:1–38, 1925. ISSN 0717-6163. doi: 10.1007/s13398-014-0173-7.2. URL https://archive.org/details/flightactivities1328lund.

[25] Edward Hieatt y Rob Mee Martin Fowler. Repository pattern. URL https://martinfowler.com/eaaCatalog/repository.html. [Online; Accedido 22-Enero-2017].

[26] University of Pennsylvania. Robotics: Estimation and learning: Gaussian mixture model (gmm), 2016. URL https:

//www.coursera.org/learn/robotics-learning/lecture/XG0WD/1-4-1-gaussian-mixture-model-gmm/. [Online; Accedido 5-Octubre-2016].

[27] OpenCV. Interfaz backgroundsubtractormog2 (background_segm.hpp), 2016. URL https://github.com/opencv/opencv/blob/master/modules/video/include/opencv2/video/background_segm.hpp. [Online; Accedido 12-Octubre-2016].

[28] OpenCV. Implementación backgroundsubtractormog2 (bgfg_gaussmix2.cpp), 2016. URL https://github.com/opencv/opencv/blob/master/modules/video/src/bgfg_gaussmix2.cpp. [Online; Accedido 12-Octubre-2016].

[29] OpenCV. Opencv background subtraction tutorial, 2016. URL http://docs.opencv.org/master/db/d5c/tutorial_py_bg_subtraction.html. [Online; Accedido 26-Septiembre-2016].

[30] OpenCV. Color conversions, 2016. URL http://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html. [Online; Accedido 18-Octubre-2016].

[31] OpenCV. Contours in opencv, 2016. URL http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_contours/py_table_of_contents_contours/py_table_of_contents_contours.html. [Online; Accedido 19-Octubre-2016].

[32] OpenCV. Structural analysis and shape descriptors, 2016. URL docs.opencv.org/3.0-beta/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html. [Online; Accedido 19-Octubre-2016].

[33] OpenCV. cv::backgroundsubtractormog2 class reference, 2016. URL http://docs.opencv.org/3.1.0/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html. [Online; Accedido 12-Octubre-2016].

[34] M H Struye, H J Mortier, G Arnold, C Miniggio, and R Borneck. Microprocessor-controlled monitoring of honeybee flight activity at the hive entrance. *Apidologie*, 25(4):384–395, 1994. ISSN 0044-8435. doi: 10.1051/apido:19940405. URL http://cat.inist.fr/?aModele=afficheN{&}cpsidt=4192550https://hal.archives-ouvertes.fr/hal-00891170/documenthttp://www.apidologie.org/10.1051/apido:19940405.

[35] Rahman Tashakkori and Ahmad Ghadiri. Image processing for honey bee hive health monitoring. In *SoutheastCon 2015*, pages 1–7,

Fort Lauderdale, FL, apr 2015. IEEE. ISBN 978-1-4673-7300-5. doi: 10.1109/SECON.2015.7133029. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7133029.

[36] Wikipedia. Background subtraction — wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/w/index.php?title=Background_subtraction&oldid=741230478. [Online; Accedido 23-Septiembre-2016].

[37] Wikipedia. Gaussian blur — wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/w/index.php?title=Gaussian_blur&oldid=739396767. [Online; Accedido 18-Octubre-2016].

[38] Wikipedia. Grayscale — wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/w/index.php?title=Grayscale&oldid=742147487. [Online; Accedido 18-Octubre-2016].

[39] Wikipedia. Scrum (software development) — wikipedia, the free encyclopedia, 2017. URL https://en.wikipedia.org/w/index.php?title=Scrum_(software_development)&oldid=761729658. [Online; Accedido 26-Enero-2017].

[40] Wikipedia. Test-driven development — wikipedia, the free encyclopedia, 2017. URL https://en.wikipedia.org/w/index.php?title=Test-driven_development&oldid=758403369. [Online; Accedido 26-Enero-2017].

[41] Li Yao and Miaogen Ling. An Improved Mixture-of-Gaussians Background Model with Frame Difference and Blob Tracking in Video Stream. *The Scientific World Journal*, 2014, April 2014. ISSN 2356-6140. doi: 10.1155/2014/424050. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4000660/.

[42] Zoran Zivkovic. Improved Adaptive Gaussian Mixture Model for Background Subtraction. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 2 - Volume 02*, ICPR '04, pages 28–31, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 978-0-7695-2128-2. doi: 10.1109/ICPR.2004.479. URL http://dx.doi.org/10.1109/ICPR.2004.479.

[43] Zoran Zivkovic and Ferdinand van der Heijden. Efficient Adaptive Density Estimation Per Image Pixel for the Task of Background Subtraction. *Pattern Recogn. Lett.*, 27(7):773–780, May 2006. ISSN 0167-8655. doi: 10.1016/j.patrec.2005.11.005. URL http://dx.doi.org/10.1016/j.patrec.2005.11.005.