UNIVERSIDAD SAN PABLO - CEU

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FINAL DE CARRERA

AUTOMATIZACIÓN DE LAS MEDIDAS DE PREFERENCIA CONDICIONADA A LOS LUGARES EN ESTUDIOS DE DESINTOXICACIÓN A COCAÍNA EN ANIMALES DE LABORATORIO

Autor: Álvaro Martínez – Leiva Rodríguez

Director: Carlos Óscar Sánchez Sorzano

Julio de 2008

UNIVERSIDAD SAN PABLO-CEU

ESCUELA POLITÉCNICA SUPERIOR

División de Ingeniería Informática y de Telecomunicación

Calificación del Proyecto Fin de Carrera

Datos personales del alumno

D.N.I.	
APELLIDOS	NOMBRE

Directores

Director 1 (tantos como sean los directores)

D./D.ª

Tribunal calificador

Presidente

D./D.ª	FIRMA

Secretario

D./D.ª	FIRMA

Vocal

D./D.ª	FIRMA

Fecha de calificación

Calificación

RESUMEN

La memoria aquí presentada describe el funcionamiento e implementación del programa de seguimiento de objetos por video (*videotracking*) desarrollado como proyecto fin de carrera.

Este proyecto surge como una necesidad real del laboratorio del Departamento de Toxicología de la Universidad San Pablo CEU en el cual se investiga con roedores mediante el sistema de preferencia condicionada a los lugares el uso de distintas sustancias para combatir la intoxicación a drogas como la cocaína.

Para dar respuesta a esta necesidad se ha diseñado un programa desarrollado en Java que automatiza el proceso de estudio del efecto reforzador de las drogas de abuso mediante la técnica de condicionamiento espacial (CPP). En particular se ha conseguido la identificación correcta del roedor en un 95% del tiempo de un estudio estándar, que comprende 30 minutos.

El programa desarrollado hace un uso intenso de procesado digital de señales, además de diferentes algoritmos propios implementados para analizar las imágenes obtenidas mediante una cámara de bajo coste y determinar la posición del animal.

ABSTRACT

This memory describes the workflow and implementation of a videotracking system developed as the Master Thesis of the Engineer telecommunications career.

This project comes as a real need of the laboratory of the Department of Toxicology of the San Pablo CEU University in which research with rodents through the preference system conditional on the sites using various substances to combat food poisoning to drugs such as cocaine.

To respond to this need a program has been developed in Java which automates the process of studying the effect of drugs reinforcement through the technique of Conditional Place Preference (CPP). It has been achieved the proper identification of the rodent in 95% of the time of a standard survey, covering 30 minutes.

The program developed makes an intense use of digital signal processing, besides various algorithms designed specifically for it to analyze the images obtained by a low cost camera and to detect animal position on the image.

A mis padres

ÍNDICES

Introducción……4

Antecedentes y estado……..6

Planteamiento y alcances……..11

Metodología experimental para el estudio de las drogodependencias……..14

Estudio de la tolerancia……15

Estudio de la sensibilización……16

Estudio de la dependencia física……..17

Estudio del efecto reforzador……...18

Autoadministración de drogas……19

Autoestimulación eléctrica intracraneal……...22

Condicionamiento espacial……..24

Protocolo del condicionamiento espacial utilizado……..25

Habitáculos……29

Entorno de trabajo……..……...33

ImageJ……..34

Image Pro Plus……35

Pax-it……36

API de la interfaz……37

AWT……37

Swing……...38

Interfaz gráfica……..39

Seleccionar fuente……40

Imagen media……...42

ROI……...44

Rata……..46

Cámara……48

Video……49

Opciones de la imagen…….50

Development……51

Velocidad del animal en el tiempo……..53

Recientes……..54

Adquisición de la fuente……57

JMF……..57

Obtener imagen de grises del reproductor……...61

Estructura del programa……...……...65

Usabilidad……...……68

Algoritmo de análisis……...……...70

Algoritmo de funcionamiento……..71

Análisis de la imagen……...76

Binarización……79

Definición de umbralización……80

Restar imagen original con la imagen media……...83

Aplicar filtro paso banda……..86

Cálculo del umbral……...90

Otsu……..92

Supervised range constrained fuzzy-C partition……..94

Detectar animal en la imagen binarizada……...…….98

Conectividad……...99

Procesar la posición……...102

Elección de objetos……104

Determinar posición exacta del animal……..108

Protección frente a errores…….111

Actualizar parámetros de detección……...114

Re-análisis de las imágenes problemáticas……116

ROIs afectados……...118

Análisis de resultados – Futuros objetivos……121

Conclusiones……...……..122

Bibliografía……...……124

Anexo 1 – Añadir dispositivos a JMF…….…...126

INTRODUCCIÓN

Los estudios de toxicidad constituyen hoy día una parte muy importante dentro del desarrollo de un nuevo fármaco y se extienden prácticamente a lo largo de todo el mismo. El objetivo de los mismos es "evaluar el riesgo o peligro potencial que un agente químico o físico puede ocasionar sobre la salud humana cuando es objeto de exposiciones agudas o crónicas". Y el estudio no se limita sólo a los fármacos, sino también a otro tipo de sustancias, como la cocaína. [28]

Para los estudios toxicológicos es muy común el uso de animales de especies adecuadas para determinar los efectos adversos en condiciones controladas, aunque a veces es difícil extrapolar los resultados de los animales al hombre. Por ello es tan importante la elección del animal a estudiar.

Tanto los humanos como los roedores evolucionaron de ancestros mamíferos comunes[14]. Como el consumo de droga se transforma en un refuerzo positivo que está mediado por estructuras cerebrales subcorticales que se conservaron durante la evolución, los estudios con roedores se utilizan para analizar las bases neurobiológicas del consumo de drogas y los procesos de refuerzo de droga en humanos. En este sentido se han hallado numerosas analogías con comportamientos humanos complejos, como por ejemplo el deseo compulsivo por el consumo, la recaída y la pérdida de control. [21]

Por ello se decidió habilitar un laboratorio en el Departamento de Toxicología de la Universidad San Pablo CEU en la cual investigar con roedores el uso de distintas sustancias para combatir la intoxicación a drogas como la cocaína.

El método de medición del nivel de intoxicación / desintoxicación es llamado CPP (*Conditioned Place Preference*) o Preferencia Condicionada a los Lugares. Con este método se miden los tiempos que pasa el animal en diferentes estancias, para luego determinar la intoxicación.

Para ello, en este laboratorio se instaló un complejo y caro sistema de seguimiento de animales por video que en unos pocos años quedó obsoleto debido al nulo aprovechamiento de técnicas de tratamiento digital de la señal.

El no procesar la señal implica que para cada experimento es necesario contar con una o varias personas para supervisar el tiempo que el animal está en cada estancia manualmente, ya que este sistema falla en gran medida.

Por todo esto se decidió aprovechar lo aprendido en técnicas de tratamiento digital de señal para implementar desde cero un nuevo programa que realice el seguimiento del animal.

ANTECEDENTES Y ESTADO

El seguimiento de animales en entornos cerrados mediante dispositivos electrónicos ha sido ampliamente estudiado anteriormente, pero la aparición de las cámaras digitales y la proliferación de algoritmos de análisis digital de imágenes digitalmente ha supuesto un gran cambio en la efectividad y sobre todo en la metodología.

Anteriormente, los sistemas electrónicos que hacían el seguimiento de un animal se basaban en sensores situados estratégicamente que medían la presencia o no del animal sobre el propio sensor.

Aparato dotado de sensores usado en experimentos con CPP

La posición del animal se monitoriza mediante una serie de diodos emisores de luz infrarroja y unos array de fotoceldas, de tal manera que cuando el animal se sitúa

encima del diodo emisor el array no capta la señal y determina que el animal se encuentra encima de él.

Este sistema es muy sencillo a la par que efectivo, ya que los sensores raramente van a tener un comportamiento erróneo (a menos que haya un problema de funcionamiento), por tanto podemos pensar que estos métodos son los más adecuados.

Sin embargo, a pesar de tener ciertas ventajas, tienen bastantes desventajas, sobre todo si lo comparamos con métodos más novedosos. El primer problema que encontramos es el encarecimiento de los habitáculos. Cambiar un laberinto (un laberinto está formado por varios habitáculos) no va a ser trivial, ya que conlleva un proceso previo de instalación de sensores.

Otro problema a destacar es el coste de estos sensores, que se multiplica cuando el experimento requiere de varios habitáculos, y que puede hacer obsoleto el sistema de medición anterior. Es decir, si anteriormente estábamos usando un dispositivo con 12 entradas para los sensores, y ahora necesitamos 16 entradas, tendremos que desechar el anterior sistema de monitorización, lo que encarece enormemente cualquier cambio en la estructura del experimento.

Pero hay más problemas asociados a este sistema, y que no se sabía hasta la llegada de los modernos sistemas de tratamiento digital de señales. El análisis de los datos obtenidos. Es evidente la pérdida de información en los sistemas donde sólo se nos informa donde está el animal. En estos no es posible hacer un seguimiento de parámetros tales como trayectoria, velocidad, torsión, etc.

Los nuevos métodos de seguimiento de animales son más sofisticados, aunque requieren de un ordenador, una cámara y el software necesario para funcionar. Quizás el coste puede parecer mayor, pero gracias al software que hemos desarrollado se puede utilizar un ordenador con pocas prestaciones (barato), una cámara web convencional (más que suficiente para nuestro fin) tampoco es muy cara, y el software se basa en herramientas Open Source que no tienen ningún coste.

Además, como veremos más adelante, cambiar el número de habitáculos o distribución de los mismos en un laberinto no influye en el sistema, lo que hace que este sea totalmente flexible.

Sin embargo, en los sistemas de seguimiento de animales mediante video ha habido una gran evolución, y nos encontramos con sistemas antiguos y prácticamente obsoletos debido al nulo uso del tratamiento digital de la señal, como en el caso del sistema instalado actualmente en la Faculta de Farmacia de la universidad.

La Universidad San Pablo CEU dispone de un Departamento de Toxicología en el que se habilitó hace unos años un CPP. Para ello dispusieron una sala de 8 metros cuadrados con la iluminación adecuada y aislada acústicamente del exterior; y en ella instalaron un sistema automático de seguimiento de objetos de una empresa especializada. El sistema está compuesto de: una cámara fija en el interior de la habitación, que desde el techo enfoca al centro de la sala; una pequeña pantalla para ver lo que está pasando en el interior de la habitación; un conversor analógico a digital para poder conectar la cámara a un ordenador; un ordenador con el software propietario instalado con el cual capturar los movimientos del animal.

Este sistema tiene una serie de desventajas para el usuario. En primer lugar, el software, como ya se ha comentado anteriormente, es propietario, por tanto no se conoce su código fuente. Al no conocer este código fuente el usuario no puede mejorar el programa, sino pagar a la empresa por nuevas actualizaciones, que a veces solo corrigen errores, no dan la funcionalidad que el usuario requiere. En segundo lugar, el sistema operativo usado por el programa es MS-DOS, lo que confiere una interacción con el usuario limitada, y el uso necesario de un ordenador exclusivamente para este fin.

El uso de este sistema operativo proporciona una interfaz gráfica limitada y poco amigable. Un ejemplo de ello es la manera de introducir en el sistema la posición de los habitáculos. Para ello, el usuario tiene que utilizar una vara metálica proporcionada por la empresa, con una pequeña bombilla en un extremo y un interruptor en el otro extremo. Con esta vara, y tras indicar al sistema que va a realizar estos pasos, el usuario enciende la bombilla sobre las esquinas de cada habitáculo, y el programa une estos puntos. Este sistema es bastante limitado, y se pierde mucho tiempo en manejarlo, por

lo que los encargados del estudio optaron por fijar donde se encontraban los habitáculos para no tener que realizar la marcación cada vez que se introducía un animal.

Sin embargo, el mayor inconveniente de este sistema es la detección del animal. Aunque no conocemos el algoritmo utilizado ya que no tenemos las fuentes, sí sabemos que no se utiliza ningún tipo de procesado de señal, sino que basa la búsqueda del animal en el brillo producido por éste. Esto es, se sabe de antemano que el animal va a ser más blanco que el fondo. El usuario tiene el control del contraste en el conversor A/D, por lo que, mientras ve el interior de la sala en la pantalla, selecciona el contraste adecuado para que la rata brille más que el resto, y de esta manera el programa capta el movimiento.

El problema de este sistema es que todo lo que brille a partir de un determinado nivel, se va a suponer que es un animal. Por tanto es necesario controlar el nivel de contraste continuamente, así como supervisar durante todo el estudio que no haya situaciones en las que el brillo no proceda del animal. Esto ocurre por ejemplo cuando el animal micciona durante la sucesión del estudio. En estos casos, la orina producida por el animal tiene un nivel de brillo igual o mayor que el brillo del propio animal, lo que conlleva a una mala detección por parte del programa. Cuando esto pasa, es necesario limpiar la estancia y volver a introducir el animal el tiempo necesario, lo que cambia los resultados del estudio. Y esto sin contar el tiempo perdido en este proceso.

Estos problemas llevaron a los encargados del CPP a supervisar manualmente el programa, midiendo los tiempos de estancia en cada habitáculo manualmente, y comprobando después con el sistema que no hubiese mucha variación. Esto conlleva una gran pérdida de tiempo para el encargado del estudio, además del gasto de la inversión que se hizo en un sistema obsoleto.

Aparte de estos problemas, la información que proporciona el sistema es limitada, ya que solo informa de los tiempos de estancia y el tiempo total, sin hacer ningún cálculo de trayectorias o de otro tipo, por lo que se pierde mucha de la información contenida en el estudio.

Por todos los inconvenientes que tiene el programa instalado, surgió la idea de crear otro nuevo, usando nuevas técnicas de procesado de imagen que no se habían estudiado a la hora de crear el sistema implantado, y que proporcionará más fiabilidad al

estudio, así como la auténtica automatización del proceso (que no necesite ningún tipo de supervisión).

Antes de implementar nuestro propio algoritmo se buscaron programas existentes que nos sirvieran de base para luego añadir funcionalidades. Para ello encontramos algunos programas útiles, como el realizado por el profesor Oswaldo Trelles con la colaboración de más investigadores, en el que realiza el seguimiento automático de un animal en un entorno conocido como Morris Water Maze. Este estudio se realiza en una piscina circular, por lo que no nos sirve para nuestro propósito, ya que nosotros no vamos a trabajar en un entorno parecido.

Existen más trabajos en entornos parecidos al nuestro, pero no hemos podido hacer uso de ellos ya que los algoritmos no se encuentran disponibles públicamente, o se han desarrollado para un dispositivo hardware en particular, lo que no concuerda con la ideología que hemos planteado, donde el programa debe funcionar con una gran gama de cámaras del mercado.

PLANTEAMIENTO Y ALCANCES

Después de ver los problemas de los que adolece el sistema propietario ya instalado es evidente que uno de los principales objetivos con los que se plantea este proyecto es la idea de desarrollar el programa íntegramente con mentalidad *Open Source*, de tal manera que el uso de herramientas libres como *ImageJ* (con la posibilidad de implementar macros o *plugins* para él) fue el primer pilar de la filosofía de este proyecto.

El uso de herramientas *Open Source* no tendría sentido si luego nos apoyáramos en dispositivos hardware restrictivos que necesitaran de drivers propietarios o programas específicos para realizar la entrada / salida de datos. Por ello se optó de nuevo por el uso de cámaras web convencionales y de bajo coste, que a pesar de no ofrecer la calidad de imagen que pueden ofrecer otras soluciones más costosas que encontremos en el mercado, sí que con la ayuda del tratamiento digital de señal nos servirá perfectamente para nuestros fines.

Y siguiendo con el mismo planteamiento, se decidió que el *PC* que se encargaría de ejecutar el programa no necesitara de igual manera de grandes prestaciones, y que usando un producto de coste relativamente bajo (incluso un *PC* desechado de alguna tarea más costosa en términos de potencia) se pudiera realizar la tarea de seguimiento de animales.

Por todo ello, el planteamiento fundamental del proyecto es realizar un programa basado en herramientas *Open Source*, que use drivers no propietarios para interactuar con los dispositivos, o al menos programas también *Open Source* que nos permitan la interacción; y por último el desarrollo de algoritmos potencialmente eficientes y que nos permitan la posibilidad de ejecución en *PC*s de bajo coste.

El objetivo principal del proyecto es la medición del tiempo que pasa en cada estancia el animal durante el transcurso del experimento de desintoxicación. Sin embargo, y sin olvidarnos de este objetivo, hay otras cuestiones, que a pesar de ser secundarias tienen un gran peso en la implementación del programa.

En primer lugar la modularidad del proyecto. Una herramienta de este tipo, y que utiliza conceptos de tratamiento digital de señal tiene que ser capaz de adaptarse a los nuevos avances en este campo. Pero no sólo es necesaria la modularidad para adaptar nuevos conocimientos, sino también para poder cambiar elementos como la cámara, la interfaz o incluso cualquier algoritmo secundario (por ejemplo la determinación del área que ocupa el animal en la imagen a partir del peso u otra característica propia del animal).

No menos importante es la optimización de algoritmos. Aunque en primer lugar se pensó en trabajar con videos almacenados, se ha desarrollado la posibilidad de ejecutar el análisis en tiempo real. Para ello ha sido necesario hacer un rediseño de la mayoría de los algoritmos para optimizar los tiempos de procesado de la imagen y determinación del objeto en la imagen. De nuevo este rediseño ha sido fácilmente realizable gracias a la idea inicial de modularidad con la que se inició el proyecto.

Pero también hay que mencionar otras funcionalidades del programa que más adelante explicaré en profundidad. Una de ellas es la medición de la velocidad del animal con respecto al tiempo, mostrando los datos en una gráfica posteriormente al análisis del experimento. También se ha añadido un seguimiento de la trayectoria del animal, de tal manera que podemos saber en todo momento los movimientos anteriores y posteriores que ha seguido nuestro sujeto.

Otra funcionalidad que se ha implementado para favorecer la usabilidad del programa, y que no influye para nada en el objetivo principal, ha sido la restauración de las últimas opciones con las que se configuró el programa.

Y por último, aunque se escapa del alcance de este proyecto, el programa se ha dejado preparado para realizar futuras implementaciones. Una de ellas es el análisis de las trayectorias anteriormente almacenadas para buscar patrones repetitivos en el comportamiento de los animales y que puedan establecer similitudes en el comportamiento que quizás lleguen al desarrollo de fármacos más efectivos. En esta misma línea es interesante el estudio del grado de torsión del animal y compararlo con el tiempo total del experimento de manera que se pueda establecer una relación entre el tiempo que el animal se encuentra en torsión y el grado de intoxicación del mismo.

METODOLOGÍA EXPERIMENTAL PARA EL ESTUDIO DE LAS DROGODEPENDENCIAS

La experimentación básica en el campo de las drogodependencias implica el desarrollo de modelos animales adecuados, capaces de reproducir en el laboratorio los componentes esenciales de la drogadicción humana. Para ello es necesario disponer de un sistema biológico provisto de receptores específicos para cada tipo de droga, de técnicas de detección y valoración de sus efectos farmacológicos y de protocolos de utilización de dichas sustancias que sean capaces de inducir los fenómenos implicados en el desarrollo de la drogodependencia. En este sentido, existen métodos que permiten estudiar los cambios adaptativos del organismo que aparecen con el consumo crónico de sustancias, entre las que podemos destacar: la **tolerancia**, **sensibilización**, **dependencia física** y **refuerzo**.

Además, para profundizar en el estudio de los sustratos neuronales que median estos procesos, resulta de gran utilidad la combinación dichos técnicas con **métodos neuroquímicos** (como la microdiálisis intracerebral o la voltametría), que serán detallados posteriormente. Por último, recientemente está cobrando gran interés el estudio a gran escala de los genes y las proteínas que están implicados en los efectos de las drogas, empleando para ello técnicas de **genómica** y **proteómica**.

La experimentación animal está exenta de muchos de los impedimentos éticos y técnicos propios de los estudios con humanos y ha proporcionado datos muy significativos para el conocimiento de las acciones de las drogas de abuso. Sin embargo, siempre se debe tener presente la imposibilidad de reproducir fielmente en el laboratorio los aspectos subjetivos o los aspectos ambientales asociados al consumo de la droga específicos de los humanos.

ESTUDIO DE LA TOLERANCIA

Se entiende por **tolerancia** a la disminución progresiva de los efectos de una sustancia a medida que se consume de forma reiterada, lo que obliga a incrementar la dosis para obtener el efecto inicial. El desarrollo de tolerancia a los efectos euforizantes y placenteros de las drogas determina el patrón de consumo en escalada de dosis de los adictos.

No se produce tolerancia a todas las drogas por igual, ni tan siquiera a todos los efectos de una misma droga; así por ejemplo, los consumidores de heroína desarrollan rápidamente tolerancia a los efectos euforizantes y más lentamente a otros efectos como la miosis. Incluso, para un mismo efecto, el desarrollo de tolerancia puede variar en función de la pauta de administración o de los factores ambientales que rodean el consumo de la droga (Stolerman, 1992).

Para el estudio de la tolerancia en el laboratorio se debe elegir en primer lugar el efecto que la representa (por ejemplo en la contracción de un músculo o en la ejecución de un comportamiento); dicho efecto determinará la complejidad del sistema biológico necesario para manifestarlo. Así por ejemplo, se pueden utilizar fracciones subcelulares, células en cultivo, órganos aislados o incluso el animal entero. A continuación se describen algunos ejemplos de estudios de tolerancia en modelos biológicos de diferente complejidad.

ESTUDIO DE LA SENSIBILIZACIÓN

La **sensibilización** es un fenómeno opuesto al de tolerancia. En este caso, la exposición crónica a una sustancia se traduce en un aumento progresivo del efecto que produce, por lo que cada vez son necesarias dosis más pequeñas para reproducir el efecto inicial. Como se ha comentado en el caso de la tolerancia, son muchos los factores que determinan la aparición o no de fenómenos de sensibilización (tipo de droga, efecto estudiado, pauta de administración, condiciones ambientales, etc.).

La sensibilización comportamental parece estar implicada en aspectos cruciales de la drogadicción como son el deseo, *craving,* y la búsqueda compulsiva de la droga (Robinson y Berridge, 1993). La sensibilización mas estudiada es la que se desarrolla en varios efectos comportamentales producidos por los psicoestimulantes. Así los humanos desarrollan sensibilización a la ansiedad o a la paranoia inducida por estas drogas, mientras que en animales esta sensibilización implica un aumento progresivo de la estimulación motora o de las estereotipias (Pierce y Kalivas, 1997).

La actividad motora de los animales se puede valorar utilizando ensayos como el campo abierto o el actímetro. El campo abierto es un recinto cuya base está dividida en cuadrantes, de forma que la actividad del animal se cuantifica contabilizando el número de

Actímetro

cuadrantes que atraviesa durante un tiempo determinado. El actímetro se coloca bajo las jaulas de los animales y registra automáticamente el movimiento del animal, bien a través de sistemas de campos magnéticos o de células fotoeléctricas.

ESTUDIO DE LA DEPENDENCIA FÍSICA

La dependencia física es un estado fisiológico alterado que se caracteriza por la necesidad de mantener unos niveles determinados de droga en el organismo para que este conserve unas condiciones biológicas cercanas a las normales. Una vez instaurada la dependencia física a una droga, la interrupción de su consumo da lugar a un **síndrome de abstinencia** que comprende una serie de signos o síntomas de naturaleza opuesta a los producidos por una administración aguda de la droga.

En el laboratorio la evaluación del grado de dependencia se realiza comúnmente valorando la intensidad de los signos del síndrome de abstinencia. Este síndrome se genera interrumpiendo el aporte crónico de la droga ("abstinencia por supresión") o administrando antagonistas ("abstinencia precipitada") que desplazan bruscamente la droga de sus receptores. El síndrome de abstinencia por supresión es poco intenso pero duradero, por el contrario, el síndrome de abstinencia precipitado es intenso y de corta duración.

Como en el caso de los estudios de tolerancia, para estudiar la dependencia física se pueden emplear **sistemas biológicos de distinta complejidad** desde células hasta animal entero. El empleo de animales como la rata o el ratón permite estudiar el comportamiento de abstinencia, ya que muchos de los signos de abstinencia que manifiestan estos animales son idénticos a los sufridos por los humanos.

ESTUDIO DEL EFECTO REFORZADOR

Todos los agentes capaces de activar el circuito de recompensa, se consideran **reforzadores positivos**. Se consideran refuerzos positivos aquellos estímulos que incrementan la frecuencia de la conducta que conlleva a su presentación.

Estos reforzadores pueden ser naturales como la comida, el agua o el sexo. Por ejemplo, si los animales aprenden que para recibir comida deben apretar una palanca, las presiones a ésta serán cada vez más frecuentes. También pueden ser también artificiales como las drogas de abuso entre las que cabría destacar la heroína o la cocaína.

Los reforzadores positivos, ya sean naturales o artificiales son capaces de activar el circuito neuronal de recompensa, cuya base la constituye el sistema mesocorticolímbico dopaminérgico que parte del área ventral tegumental y envía proyecciones al Núcleo Accumbens.

Las drogas de abuso son capaces de crear una conducta de administración compulsiva de la sustancia de forma periódica y continuada. Esta actitud es la desencadenante de la dependía psicológica.

Cuanto mayor es la capacidad de refuerzo de una sustancia, tanto mayor es el grado de dependencia psicológica que produce. La comida es un reforzador positivo natural, si los animales aprenden que para recibir comida deben apretar una palanca, las presiones a la palanca serán frecuentes.

Las drogas de abuso actúan también como reforzadores positivos y se ha comprobado que son capaces de activar el mismo circuito cerebral que activan los reforzadores naturales, como la comida, el agua o el sexo. La base del **circuito neuronal de recompensa** la constituye el sistema mesolímbico doparminérgico que parte del área ventral tegmental y envía proyecciones al núcleo accumbens (Figura 1)

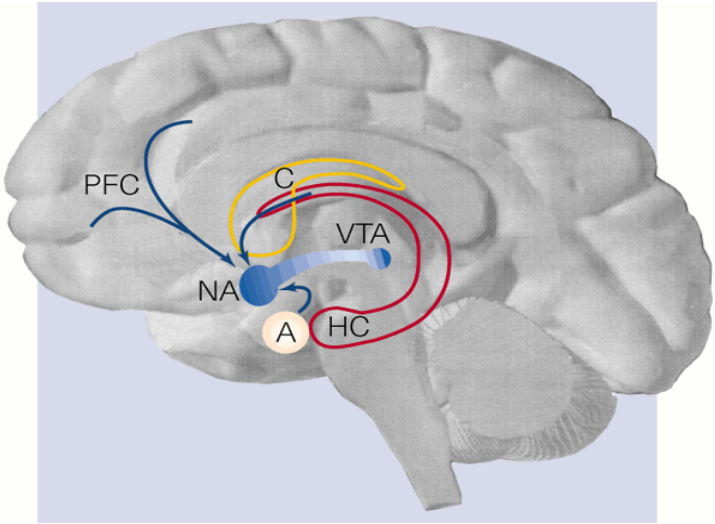


Figura 1. **Circuito cerebral de recompensa**. La base la constituye el sistema mesolímbico doparminérgico que se origina en el área ventral tegmental (VTA) y proyecta en el núcleo accumbens (NA). Se presentan además otras estructuras implicadas en el refuerzo: A (amígdala), HC: hipocampo, PFC (corteza pre-frontal), C (núcleo caudado). Tomado de Robbins y Everitt, Nature 398: 567-570, 1999.

En el laboratorio el refuerzo positivo producido por una droga se puede valorar mediante los tests de **auto administración de drogas**, **auto estimulación eléctrica intracraneal** y **condicionamiento espacial.**

AUTOADMINISTRACIÓN DE DROGAS

La autoadministración de drogas es el modelo animal más directo para estudiar la capacidad de una sustancia para actuar como reforzador positivo. Sin que sea necesario crear condiciones ambientales adversas o estresantes y sin una historia previa de tolerancia o de síndrome de abstinencia, los animales se autoadministran la mayoría de las sustancias que actúan como drogas de abuso en humanos (ver tabla adjunta), mientras que las sustancias con poco poder adictivo generalmente no dan lugar a una conducta de autoadministración en animales.

SUSTANCIA	ESPECIE
Amobarbital	rata, mono
Cafeína	rata, mono
Cocaína	gato, cerdo, rata, mono
Codeina	rata, mono
Diazepam	mono
Dietilpropión	rata, mono
Etanol	rata,mono
Heroína	rata, mono
Hexobarbital	rata
Meperidina	rata
Metadona	rata,mono
Morfina	perro, ratón, rata, mono
Nicotina	perro, rata, mono
Fenciclidina (PCP)	perro, rata, mono
Secobarbital	mono

Tabla II. Ejemplos de sustancias autoadministradas por animales por vía intravenosa

Por todo ello, la autoadministración de drogas es un modelo animal útil para predecir el potencial adictivo de una sustancia y por tanto podría proporcionar datos interesantes para dirigir la síntesis de nuevos fármacos hacia moléculas que no creen adicción en humanos. Además, frente a los modelos animales de administración pasiva de drogas, el modelo de autoadministración permite reproducir una situación más cercana a la real ya que supone un consumo voluntario de la droga; asimismo, los animales son capaces de mantener niveles estables de droga en sangre sin alcanzar los niveles tóxicos que pueden fácilmente obtenerse en una administración forzada.

Los primeros estudios experimentales sobre la autoadministración de drogas fueron realizados por Nichols y colaboradores en 1956. En dichos estudios se ofreció una solución de morfina a ratas como alternativa al agua de bebida y se demostró que el

consumo de dicha solución superaba al del agua normal, se comprobó también que no influía en este hecho el sabor de la solución.

Poco tiempo después, James Week (1962) puso a punto la metodología de la **autoadministración de drogas por vía intravenosa**; esta vía es la más utilizada en la actualidad y por ello se expondrá con detalle en los siguientes subapartados. Se han desarrollado además modelos de autoadministración de drogas por otras vías: oral, intragástrica, subcutánea, inhalatoria o intracraneal.

La estimulación eléctrica de determinadas zonas cerebrales produce recompensa tanto en animales como en humanos (Olds y Milner, 1954). Dichas zonas coinciden con las que son activadas por los reforzadores naturales (comida, sexo, etc.) que, como ya se ha comentado, son también activadas por las drogas de abuso. Con lo que es lógico suponer que las sustancias con potencial adictivo, facilitaran o potenciaran la autoestimulación eléctrica intracraneal. De hecho esta metodología fue una de las primeras que mostro que las drogas de abuso interaccionaban con el circuito neuronal de recompensa.

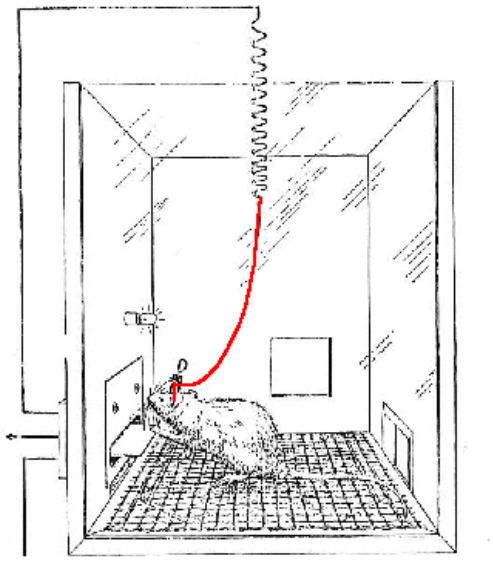
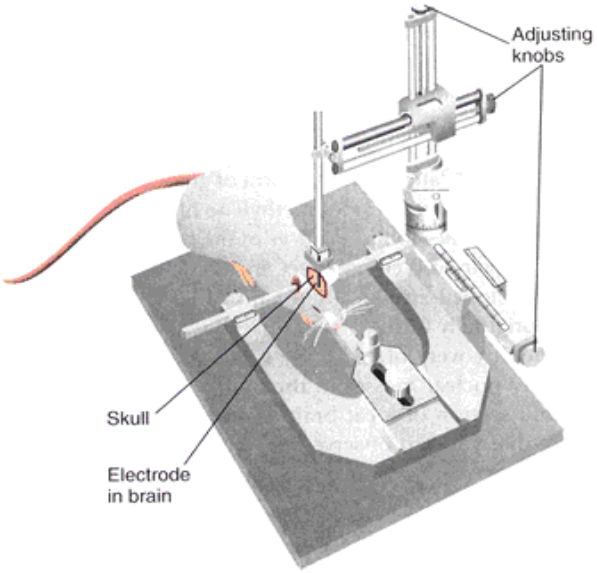


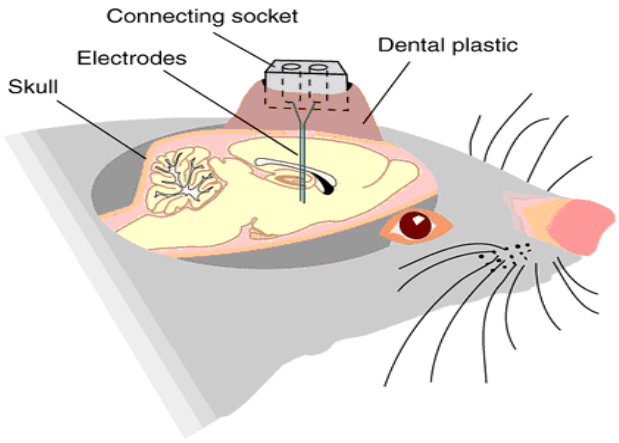
Figura 2

El protocolo básico para poder estudiar en el laboratorio la autoestimulación craneal se inicia con la colocación de un electrodo en una zona cerebral determinada. Para ello es necesario utilizar un marco estereotáxico. Este aparato permite por un lado, inmovilizar la cabeza del animal y por otro realizar movimientos finos en tres dimensiones con el electrodo para situarlo en el núcleo cerebral deseado. Los animales recibirán una pequeña descarga eléctrica a través de dicho electrodo cuando ejecuten una tarea, que consiste normalmente en presionar una palanca.



Marco estereotáxico. Este aparato se emplea para implantar electrodos, cánulas o sondas en núcleos cerebrales concretos. Los tormillos micrométricos permiten realizar movimientos finos en tres dimensiones (rostro-caudal, lateral y dorso-ventral). Los movimientos en cada dirección se realizan de acuerdo a las coordenadas que marca el atlas cerebral para cada núcleo cerebral.

Como ya se ha explicado, las drogas de abuso facilitan o potencian la estimulación eléctrica intracraneal. Esta facilitación puede manifestarse por ejemplo como un aumento de la **frecuencia** de las respuestas que ejecuta el animal (numero de presiones de palanca/min.) para una intensidad de corriente eléctrica dada o bien como una disminución del **umbral** de autoestimulación (intensidad de corriente necesaria para que el animal responda) (ver revisión de Bozarth, 1987b).



Electrodo fijado en el cerebro de la rata para ensayos de autoestimulación intracraneal.

El condicionamiento espacial es un modelo conductual en el que se valoran las propiedades motivacionales que adquieren aquellos estímulos ambientales que están asociados al consumo de una droga. Básicamente, este modelo consiste en cuantificar la preferencia o aversión de un animal por un determinado espacio en el que ha sido tratado con la droga en estudio.

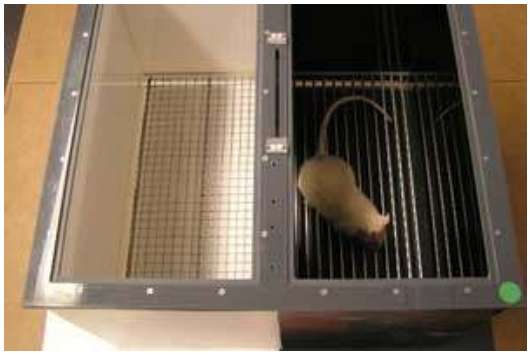


Figura 3

Los protocolos utilizados para el estudio del condicionamiento espacial son variados. Generalmente se emplean laberintos divididos en varios compartimentos fácilmente distinguibles por el animal (ver Figura 3). Durante varios días los animales son tratados con la droga (o combinaciones de drogas) en estudio en un compartimento determinado; de esta manera el animal aprende a relacionar el efecto de la droga con dicho compartimento. Posteriormente se le permite deambular libremente por el laberinto, de forma que el tiempo que pasa el animal en el compartimento en el que recibió la droga se considera proporcional al refuerzo positivo de la misma.

Se ha comprobado que drogas que producen abuso en humanos generan de forma consistente preferencia espacial en este modelo, tal es el caso de la morfina, la cocaína, el diazepam o el MDMA ("éxtasis") entre otros; lo cual garantiza la validez de este modelo para detectar las propiedades reforzadoras de nuevas sustancias. Otro aspecto que permite valorar este modelo es la capacidad de ciertas sustancias para inhibir el refuerzo producido por las drogas de abuso (ver revisión de Tzschentke, 1998); dichas sustancias se perfilan como nuevas armas terapéuticas para el tratamiento de las drogodependencias.

PROTOCOLO DEL CONDICIONAMIENTO ESPACIAL UTILIZADO

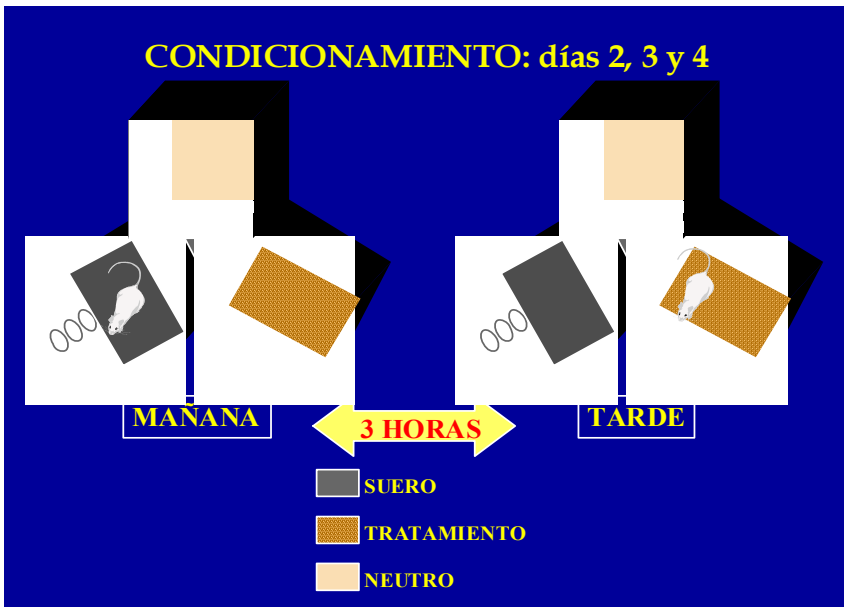
Se utiliza un laberinto de metacrilato negro formado por 3 brazos rectangulares de igual tamaño, situados a 120º uno de otro, y unidos por una zona central triangular que permite el acceso a dichos brazos mediante puertas panelables.

Cada brazo difiere en los colores o textura de su suelo y paredes para que el animal sea capaz de distinguirlos:

- Suelo de corcho y paredes negras.
- Suelo cubierto con viruta de maíz y las paredes pintadas con rayas blancas.
- Suelo de metacrilato negro y las paredes decoradas con círculos blancos.

El laberinto está instalado en el laboratorio del departamento de toxicologia de la Universidad San Pablo - CEU. Se encuentra en una habitación insonorizada, con temperatura constante e iluminación adecuada.

El procedimiento experimental consta de tres fases con distinta duración de cada una. El precondicionamiento, el condicionamiento y la libre elección. El recorrido del animal se registra a través de un sistema de *videotracking* que permite calcular automáticamente el tiempo de estancia en cada brazo para cada animal.



Esquema del protocolo de la fase de condicionamiento en el test del condicionamiento espacial.

PRECONDICIONAMIENTO: DÍA 1

Para examinar la conducta espontanea (basal) de los animales, cada uno de ellos se sitúa individualmente en el centro del laberinto desde donde tiene libre acceso a todos los brazos durante 30 min.

Los individuos que presentan una excesiva preferencia (estancia superior al 60 % de la sesión) o aversión (estancia menor del 10 % de la sesión) por alguno de los brazos son descartados.

El resto de los animales se distribuye en dos grupos de tal forma que el tiempo medio de permanencia en cada brazo sea similar en todos los grupos. Después, arbitrariamente y para cada animal, se asocia un brazo a la administración del tratamiento (droga o combinaciones de drogas en estudio), otro a la administración de salino y el tercero se considera neutro.

CONDICIONAMIENTO: DÍAS 2, 3 Y 4

Esta fase comprende tres días de sesiones dobles (una de mañana y otra de tarde). El primer día en la sesión de mañana los animales reciben una administración de salino (10 ml/kg, i.p.) y son inmediatamente confinados en el brazo asociado a salino durante 30 min.

Por la tarde los animales son tratados con el tratamiento en estudio y confinados en el brazo asociado durante otros 30 min. Se mantiene un intervalo de 3 h entre sesiones de mañana y tarde. En los días siguientes se realiza el mismo procedimiento pero el orden de los tratamientos (mañana/tarde) se alterna para evitar la influencia del ritmo circadiano.

LIBRE ELECCIÓN: DÍA 5

El ultimo día los animales vuelven a explorar libremente el laberinto en las mismas condiciones que en la fase de Precondicionamiento y se vuelve a registrar automáticamente el tiempo que cada animal permanece en cada brazo.

Es fácil deducir que si el tratamiento actúa como un reforzador positivo, el animal pasara más tiempo en el brazo asociado a dicho tratamiento que en el brazo asociado a salino, y que ocurrirá lo contrario si el tratamiento resulta aversivo. Sin embargo, antes de llegar a esa conclusión, hay que comprobar que el tratamiento no ha afectado a la integridad de los procesos de memoria y aprendizaje del animal.

En este sentido, la inclusión de un brazo neutro en nuestro laberinto, en comparación con el diseño más extendido de dos brazos, resulta sumamente útil, ya que se ha comprobado que los roedores exhiben una preferencia natural por los entornos parcialmente novedosos en condiciones experimentales semejantes a la nuestra (Carr y col., 1989; Parker y col, 1992).

De forma que el brazo neutro de nuestro aparato, solo conocido por los animales el día del Precondicionamiento y no utilizado durante la fase de condicionamiento, es el que mejor cumple este requisito. Por tanto, cabria esperar que el día de la libre elección los animales pasaran más tiempo en este entorno que en el brazo asociado a salino.

Teniendo en cuenta todo esto, el orden de preferencia debería ser:

Si el tratamiento actúa como **reforzador positivo:**

$$Tiempo_{\ brazo\ tratamiento} > Tiempo_{\ brazo\ neutro} > Tiempo_{\ brazo\ salino}$$

Si el tratamiento es **aversivo:**

$$Tiempo_{\ brazo\ neutro} > Tiempo_{\ brazo\ salino} > Tiempo_{\ brazo\ tratamiento}$$

No obstante, es siempre recomendable complementar el condicionamiento espacial con modelos experimentales específicamente diseñados para valorar los procesos de memoria y aprendizaje del animal (por ejemplo, en nuestro laboratorio empleamos el test de la evitación pasiva y el laberinto en “Y”). Las tareas de la evitación pasiva son aquellas en las que el sujeto debe inhibir una respuesta para evitar

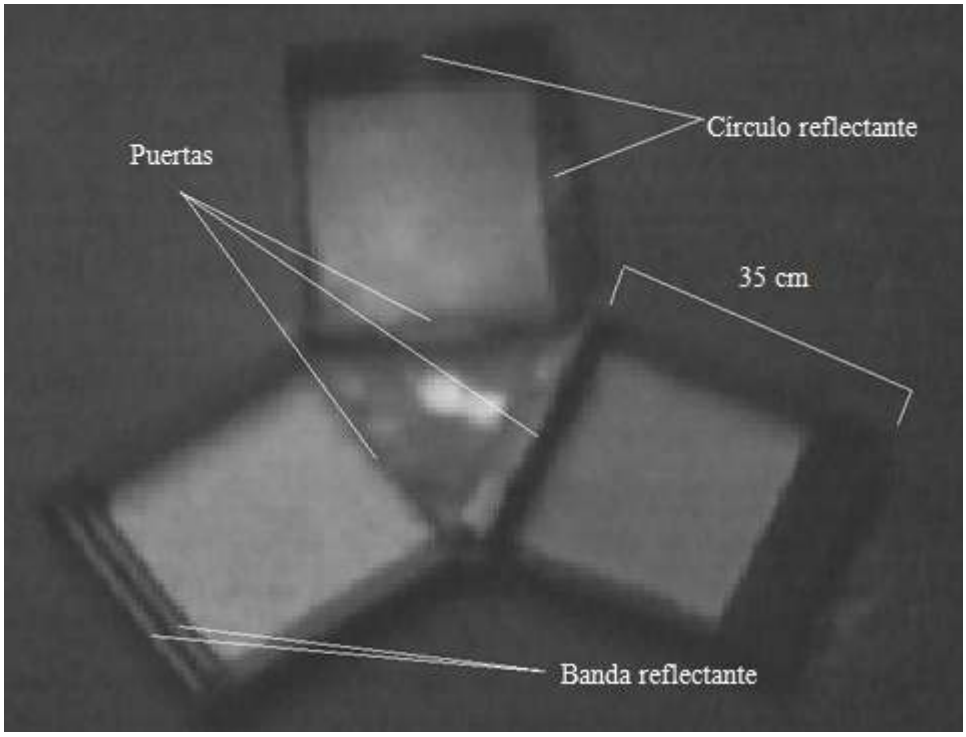
una estimulación aversiva. En la siguiente página podemos observar una imagen de un laberinto en "Y" utilizado para este fin.

Laberinto en "Y"

Además de valorar el refuerzo producido por una nueva sustancia, el condicionamiento espacial permite explorar la posibilidad de modular farmacológicamente las propiedades reforzadoras de las drogas de abuso. Por ejemplo, en nuestro laboratorio hemos comprobado que la yohimbina (un antagonista α2-adrenérgico) inhibe el desarrollo de la preferencia al sitio inducida por morfina, ya que los animales tratados con morfina y yohimbina durante el condicionamiento pasaron menos tiempo en el brazo asociado a este tratamiento que los animales tratados solo con el opiáceo.Por todo esto se decidió usar cámaras convencionales, por puerto USB, y con drivers abiertos como son las *webcams*.

LOS HABITÁCULOS

Tal como se explica en el *Protocolo del condicionamiento espacial utilizado* en la parte de CPP, se utiliza un laberinto de metacrilato negro formado por 3 brazos rectangulares de igual tamaño, situados a 120º uno de otro, y unidos por una zona central triangular que permite el acceso a dichos brazos mediante puertas panelables. Cada brazo mide treinta y cinco centímetros de lado y está abierto por arriba.



Laberinto de 3 brazos

En esta imagen se puede observar la configuración de uno de los laberintos más usados en el laboratorio. Además de las características descritas más arriba sobre el laberinto, es necesario destacar el por qué del uso de reflectantes en las paredes, además del relleno del suelo, diferente para cada habitáculo, tal como se puede observar en la fotografía.

Debido a su naturaleza, a las ratas y a los ratones no les gustan los espacios con luz, ya que prefieren situaciones de oscuridad. Con las bandas y los círculos reflectantes

se intenta dar más luz al habitáculo, de tal manera que de manera natural no se sientan atraídos por él.

Sin embargo, un componente que les gusta mucho encontrar en el habitáculo es el serrín. Por ello, el habitáculo con una mayor cantidad de luz (aquel que tiene las bandas) es a su vez el habitáculo con más serrín, y el habitáculo con menos luz es el que tiene menos serrín.

Podemos distinguir todo esto si nos fijamos en la fotografía anterior. El habitáculo con las bandas reflectantes tiene el suelo más claro (más serrín), mientras que el habitáculo de abajo a la derecha, que no tiene ningún reflectante es el que menos serrín tiene.

Con ésta configuración se intenta que el animal sienta la misma predisposición a cualquiera de los brazos, y que en cada uno de ellos encuentre algún componente aversivo, además de alguna razón por la que pasar tiempo en la estancia.

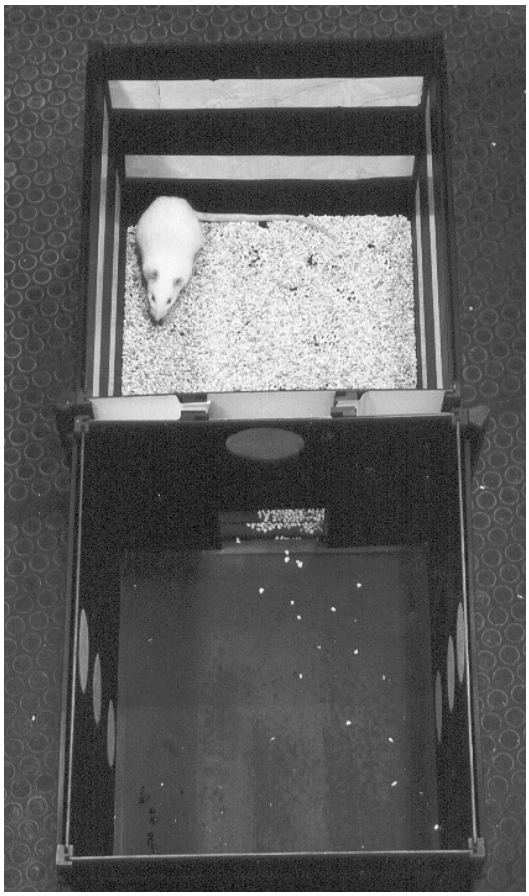
Comentar además que el serrín tiene el problema de retener olores, por lo cual es necesario cambiarlo durante el experimento, para que no influya en la decisión del animal. Es necesario también redistribuir el serrín después de cada animal pues éste lo mueve entre los brazos, y como hemos dicho antes, es necesario mantener una componente de aversión y de atracción.

Tal y como se explicó anteriormente, cuando se realiza el paso de **condicionamiento** sólo se usa un brazo, un habitáculo es asociado a suero, y el otro a la droga en estudio. Cada uno de estos brazos son elegidos aleatoriamente entre cada uno de éstos tres al inicio del experimento.

El laberinto analizado en la foto anterior es sólo uno de los posibles laberintos que se usan para el CPP, pero existen más posibles distribuciones, según qué se quiera estudiar. Una:

Laberinto de 4 brazos, dos de ellos sin paredes verticales

En esta imagen podemos observar cuatro habitáculos. Dos de ellos no tienen paredes verticales, por lo que el animal tendrá preferencia por los brazos con paredes. Sin embargo cuando se les inyecte diazepam la aversión a los habitáculos sin paredes verticales desaparecerá.



Laberinto de 2 brazos

En la figura anterior podemos observar otro tipo de laberinto, formado por dos brazos. Este laberinto es el más utilizado actualmente en el laboratorio de toxicología.

ENTORNO DE TRABAJO

Aunque este proyecto tiene algunos objetivos muy ambiciosos, la idea inicial de éste no incluía muchos de ellos (como una interfaz gráfica propia). En un principio se pensaba desarrollar un *plugin* para algún programa de análisis de imágenes no comercial, Open Source y con una gran comunidad de usuarios.

Para esta idea inicial era necesario elegir un programa de análisis de imágenes que se ajustaran a estas necesidades, pero que en caso de necesitar más funcionalidad permitiera evolucionar. He dicho que se buscaba un programa con una gran comunidad de usuarios ya que una gran comunidad de usuarios indica que el programa tiene la capacidad de hacer cosas diferentes (para adaptarse a las necesidades de todos); e incluso de crear tus propios programas para el analizador de imágenes. Una gran comunidad también implica una mayor documentación, posibles errores comentados y arreglados, además de un largo etcétera de ventajas.

Aunque la decisión fue relativamente fácil, se estudiaron varios analizadores, aunque debido a su gran popularidad y manejo nos quedamos con ImageJ. Este analizador tiene todas las funcionalidades necesarias. Se pueden generar macros (una serie de órdenes enlazadas), *scripts* más complejos con más funcionalidad e incluso plugins. ImageJ está programado íntegramente en Java, por tanto los *scripts* y los *plugins* que se desarrollan deben escribirse en este mismo lenguaje.

Java es un lenguaje orientado a objetos con una gran ventaja: Independencia del sistema operativo y plataforma. Además es robusto y tiene un gestor de procesos eficiente que permite usar procesadores de más de un núcleo.

Por tanto, el programa base elegido fue ImageJ, aunque hay otros en el mercado, principalmente los tres que analizo a continuación.

IMAGEJ

ImageJ es un programa de procesamiento de imágenes de dominio público basado en Java, e inspirado en el *NIH Image* para *Macintosh*. Se puede ejecutar tanto como un *applet* online como una aplicación descargable en cualquier ordenador con Java 1.4 o superior.

ImageJ puede mostrar, editar, analizar, procesar, salvar e imprimir 8-bit, 16-bit y 32-bit imágenes. Soporta *stacks* o pilas de imágenes en una ventana. Es multihilo, por lo que operaciones con gran consumo como lectura de imágenes pueden ser paralelizadas con otras operaciones.

Open Source: Sí, escrito en Java

Plugins: Sí, en Java. También admite scripts (javascript) y macros.

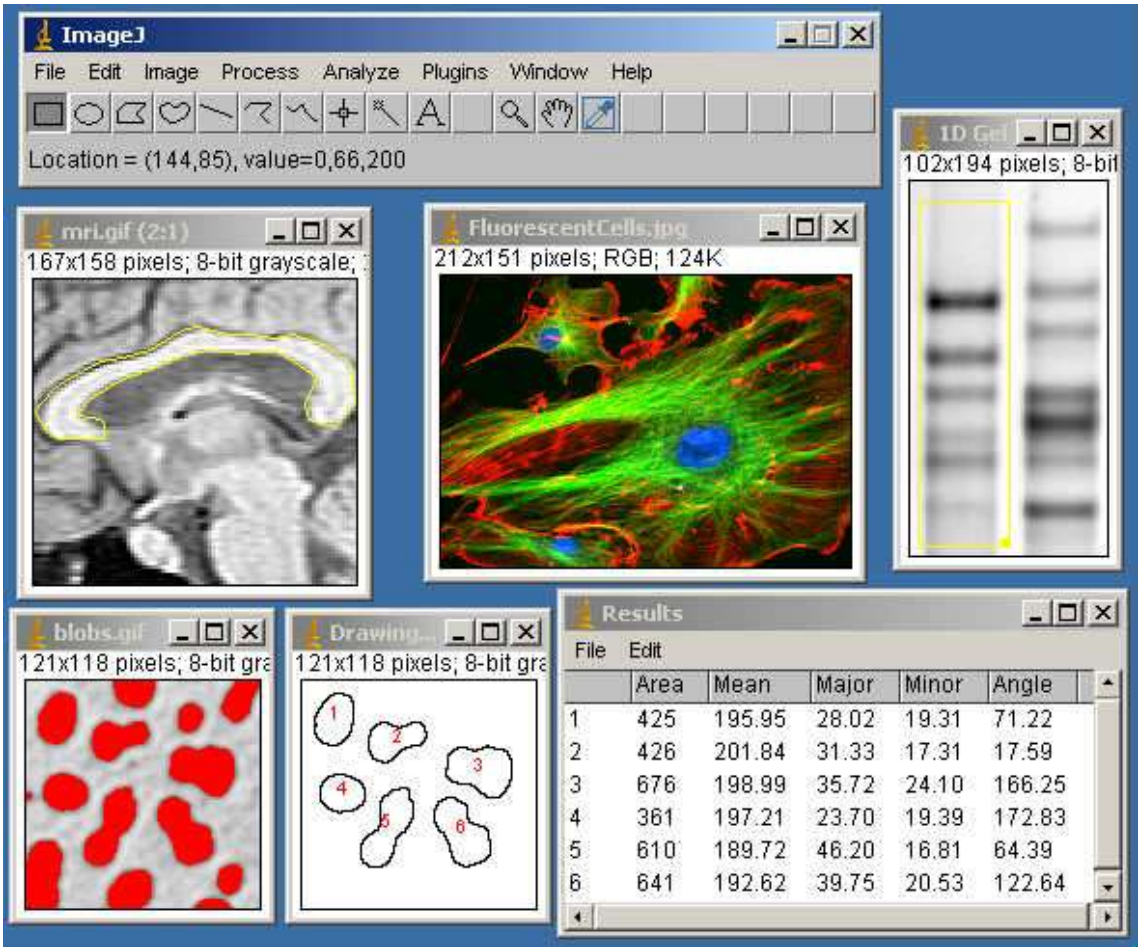
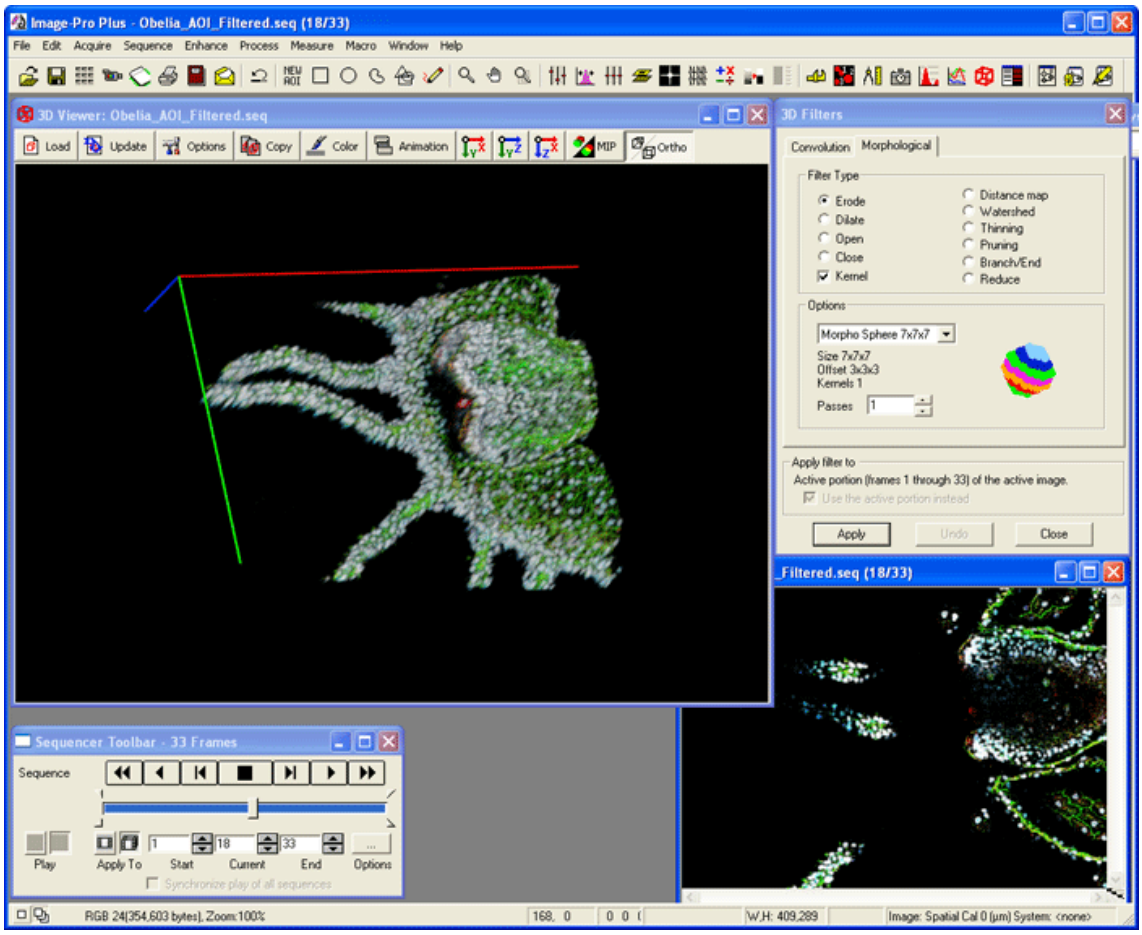


IMAGE PRO PLUS

Image Pro Plus combina las últimas herramientas de análisis de imagen para científicos o industria en un paquete completo de software intuitivo. Tras los 20 años de desarrollo, evolución y ayuda de usuarios, Image Pro Plus tiene las herramientas que necesitas para fácilmente capturar, mejorar, procesar y medir imágenes.

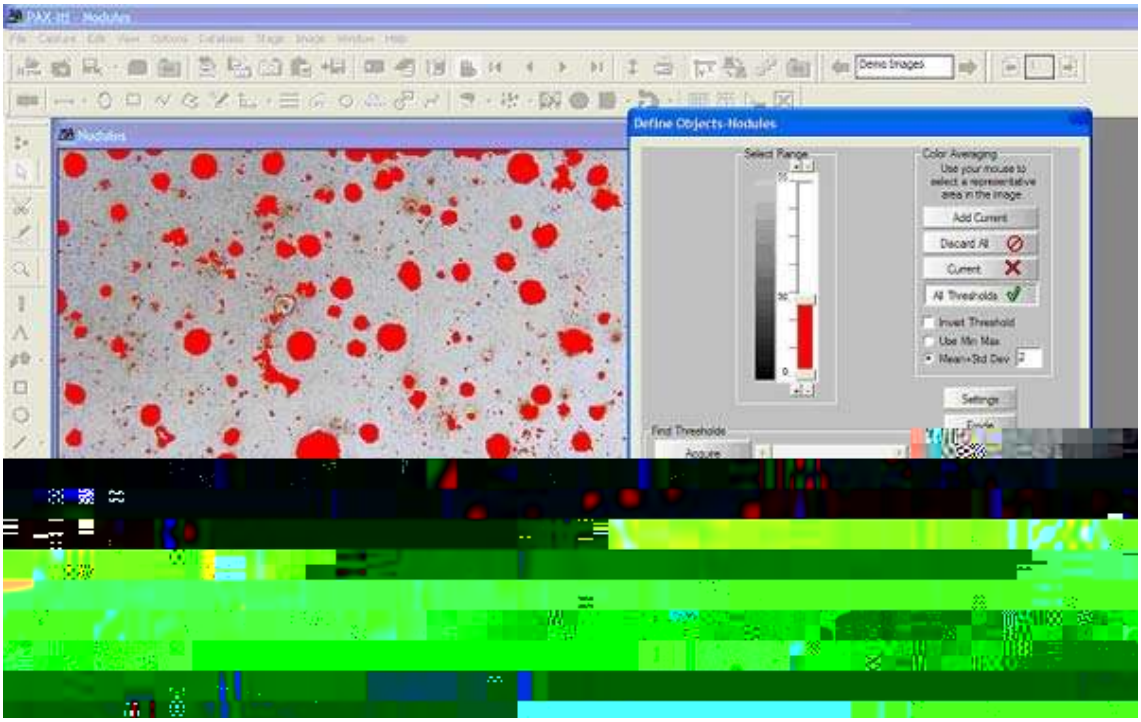
Open Source: No. MediaCybernetics.

Plugins: Sí, escritos en Java o C/C++.



PAX-IT

El *software* de análisis de imágenes PAX-it proporciona las herramientas de medida más necesitadas, alejándose de otras soluciones de análisis de imágenes más pesadas que en muchos casos solo ofrecen confusión. Todas las características de análisis se pueden ejecutar desde un asistente fácil de usar que te guiará en el proceso de analizar las imágenes. Las rutinas pueden ser más tarde salvadas para llamarlas cuando se necesiten desde un script personalizado.

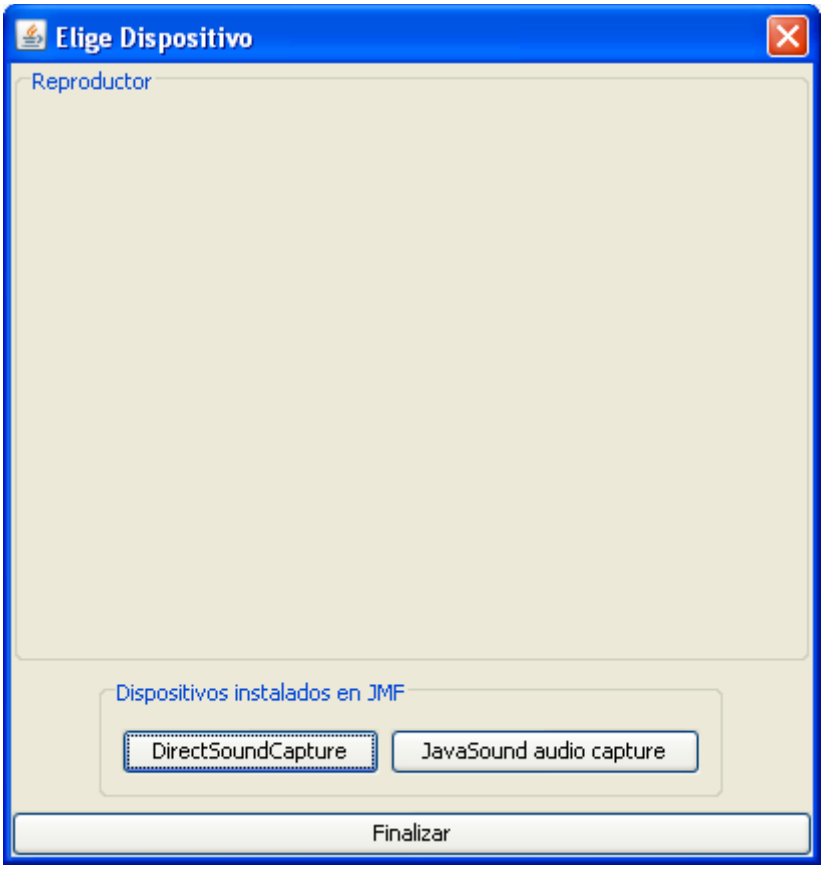


Open Source: No. MIS.

Plugins: No, scripts personalizados

SELECCIONAR FUENTE

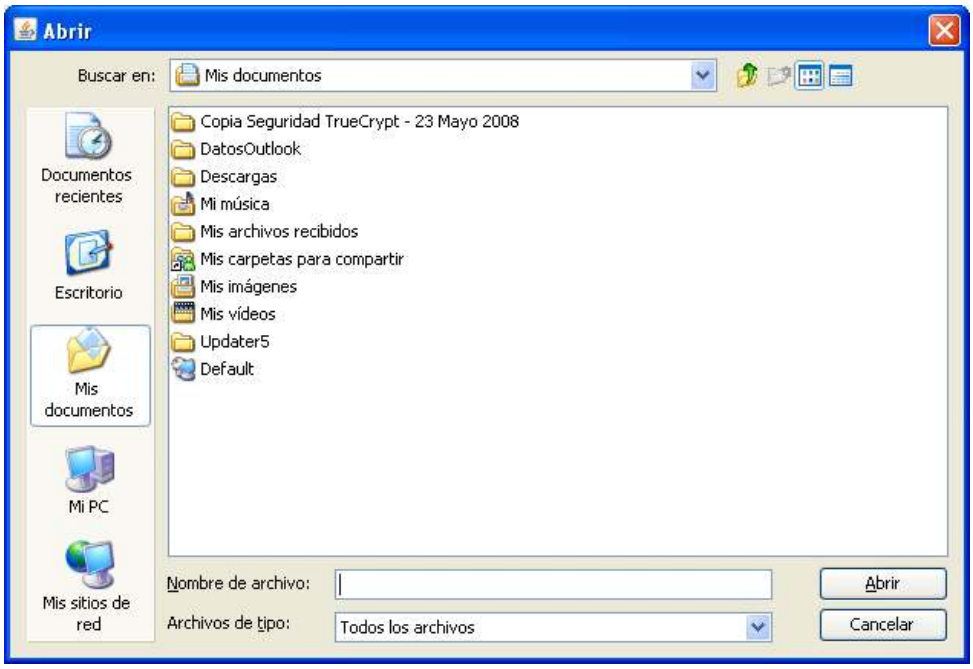
El diálogo **Archivo** contiene dos opciones: Abrir y Salir. Cuando seleccionamos Abrir podemos elegir entre Imágenes, Cámara y Video. Cuando elegimos la cámara aparecerá el siguiente diálogo:



Elegir dispositivo

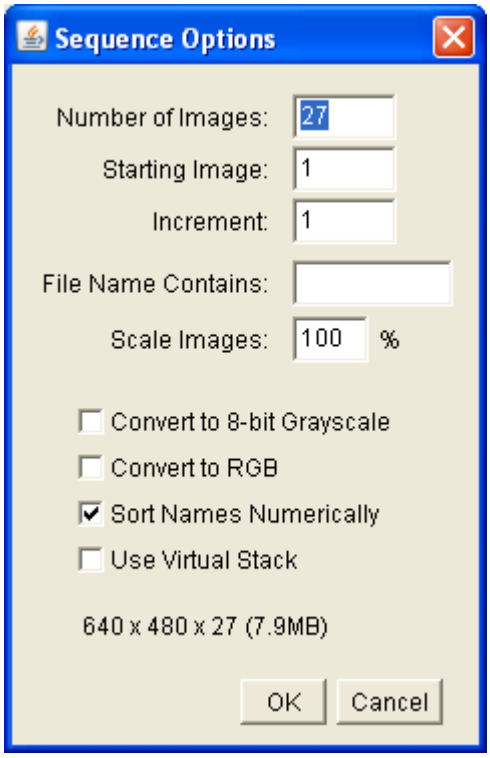
Aquí podemos elegir entre los **Dispositivos instalados en JMF**: para más información sobre instalación de dispositivos en JMF mirar en el *Anexo - 1 Añadir dispositivos a JMF*. Después de elegir el dispositivo, y antes de aceptar nos mostrará en el espacio vacío la imagen actual del video, para el caso en el cual tengamos más de un dispositivo conectado y no sepamos cuál de ellos es el adecuado.

Si elegimos abrir un video aparecerá un diálogo donde elegir el fichero a abrir, igual que si elegimos abrir imágenes:



Diálogo para abrir Videos o Imágenes

Sin embargo, en el caso de abrir imágenes, aparecerá otro diálogo, donde tendremos que elegir el número de imágenes que queremos analizar, el nombre de la primera imagen, el incremento (por si necesitamos saltar una imagen cada vez).



Diálogo para importar imágenes

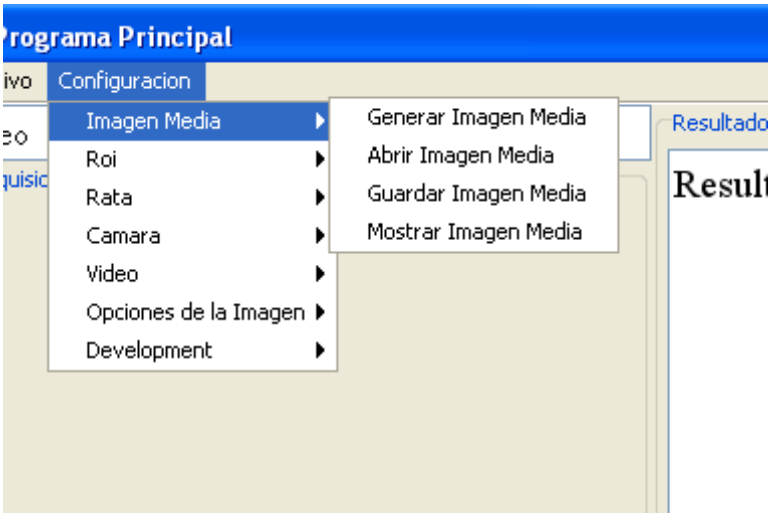
También debemos decidir si escalar las imágenes (en porcentaje), si queremos convertir las imágenes a 8-bit, o convertirlas a RGB. Las dos últimas opciones, *Sort Names Numerically* y *Use Virtual Stack* son opciones propias de ImageJ, para saber más sobre ellas:

http://rsb.info.nih.gov/ij/docs/menus/file.html - import.

El diálogo de configuración contiene las siguientes opciones: **Imagen media**, **Roi**, **Rata**, **Cámara**, **Video**, **Opciones de la imagen**, ***Development***.

IMAGEN MEDIA

Aunque más adelante veremos el uso que se le da a la imagen media con respecto al procesamiento de imágenes, en este punto aclararemos cómo se genera la imagen media y el resto de opciones asociadas.



Opciones Imagen Media

En la figura se pueden observar las opciones disponibles. **Abrir Imagen Media** permite cargar desde un fichero la imagen media. **Guardar Imagen Media** almacena en un fichero la imagen media actual. **Mostrar Imagen Media** muestra en la zona inferior izquierda de la interfaz principal la imagen media que se utiliza en la actualidad. **Generar Imagen Media** da paso a un proceso más complejo, que se inicia con el siguiente diálogo:

Generar imagen media

Aquí debemos elegir entre Imagen, Video o Cámara. Si elegimos Imagen volveremos a elegir entre el número de imágenes a cargar y el resto de *Sequence Options*, como a la hora de abrir imágenes.

Si seleccionamos generar la imagen media a partir de un video, esta opción nos llevará a un diálogo donde elegiremos el archivo desde el cual generar la imagen media.

Si elegimos la Cámara, aparecerá el siguiente diálogo, en el cual nos pedirá cuanto tiempo queremos que se grabe video para generar la imagen media:

Generar imagen media a partir de la cámara

El proceso para generar una imagen media parte de una secuencia de imágenes, que se pueden generar desde una cámara, desde un video, o tenerlas guardadas en disco. Después de tener las imágenes en una pila, ejecutamos el siguiente proceso:

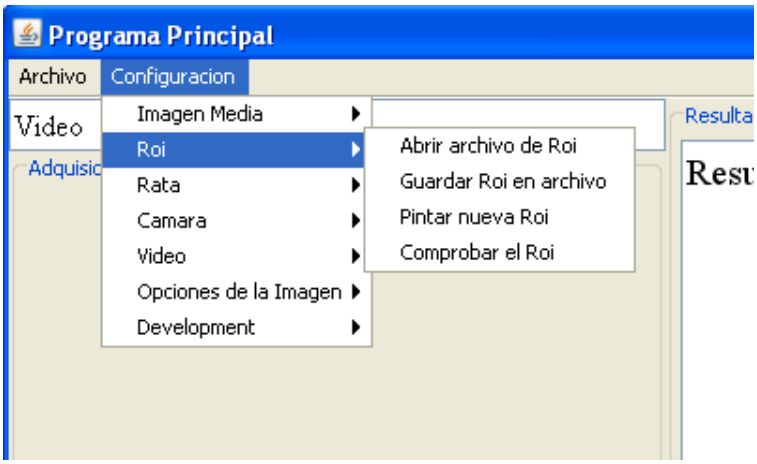
```
for (int s=1; s<imgStack.getSize(); s++)
{
   ip = imgStack.getProcessor(s); // Obtener imagen número s
   for (int x=0 ; x<width; x++)
      for (int y=0; y<height; y++)
         if(RoiTools.isInRoi(x,y,vRoi)) // Si el punto pertenece al ROI
            arrayAverage[x][y] = arrayAverage[x][y] + ip.getPixel(x,y);
}
for (int x=0 ; x<width; x++)
   for (int y=0; y<height; y++)
      if(RoiTools.isInRoi(x,y,vRoi))
         ipAverage.putPixel(x,y, arrayAverage[x][y]/imgStack.getSize() );
```

Se suman todos los pixeles de cada imagen de la secuencia en un nuevo array, y luego antes de guardarlos en una nueva imagen dividimos ese valor por el número de imágenes en la pila.

Tras este proceso se muestra en la interfaz principal la imagen generada, y podemos guardarla si así lo deseamos, o empezar el análisis con ella.

ROI (*REGION OF INTEREST*)

El ROI o región de interés define las zonas que se van a procesar en una imagen. En el caso del CPP, el ROI es el laberinto donde tenemos al animal. En el proceso para generar la imagen media usamos la función *RoiTools.isInRoi(x,y,vRoi)*. Esta función determina si el punto pertenece al ROI o no, con lo decidimos si procesar o no el punto.



Opciones ROI

Este submenú tiene las mismas opciones que el submenú Imagen Media. Podemos **Abrir archivo de Roi**, donde elegimos un archivo guardado previamente. También podemos **Guardar Roi en archivo**, y **Comprobar el Roi**, donde podemos mostrar la imagen actual con el ROI superpuesto en ella.

La siguiente opción a destacar es **Pintar nueva Roi**. Cuando seleccionamos esta opción nos aparece un nuevo diálogo donde seleccionaremos cada uno de los habitáculos que componen el laberinto.

Cada habitáculo será un polígono. Lo primero que hay que hacer es seleccionar la casilla Nuevo Polígono. Una vez hecho esto haremos click en cada una de las esquinas del habitáculo, y éstas se dibujarán en la imagen para tener un mejor control del proceso. Cuando hayamos terminado de definir el polígono seleccionaremos la casilla Guardar Polígono.

Tras esto se pintará el polígono en la imagen, y aparecerá un *checkbox* para cada Roi. Más adelante se explicará en profundidad a qué afecta seleccionar esta casilla y el por qué de ella, pero como introducción saber que sirve para aquellos habitáculos en los

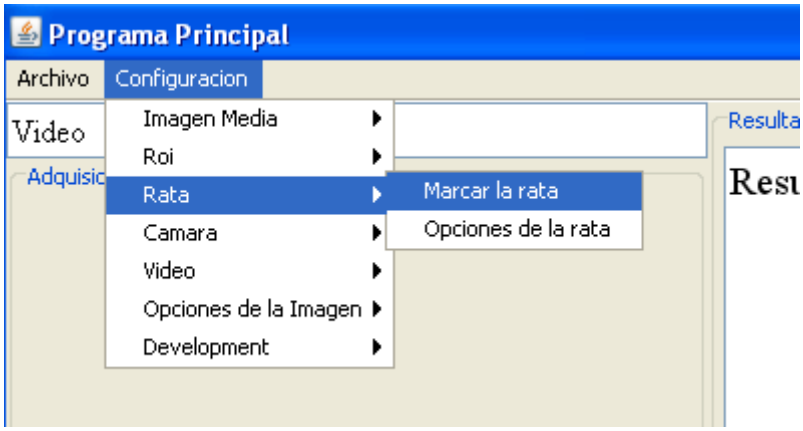
que el serrín tiene un color más blanco que el animal. Por ello, en la imagen que obtenemos de la cámara el fondo (serrín) es más claro que el animal, y a la hora de usar la imagen media es necesario hacerlo de otra manera.

El diálogo del que estamos hablando corresponde a la siguiente figura. En ella se puede ver el proceso de determinar los 4 ROIs. El primero ya se ha guardado, por lo que nos aparece pintado en la imagen, y también aparece el *checkbox* que indica que el fondo es más claro que el animal.

Diálogo para Pintar el Roi

Cuando hayamos terminado de pintar el ROI, haremos click en Aplica Roi. Esto almacena todos los polígonos para usarlos posteriormente en el análisis de las imágenes.

RATA



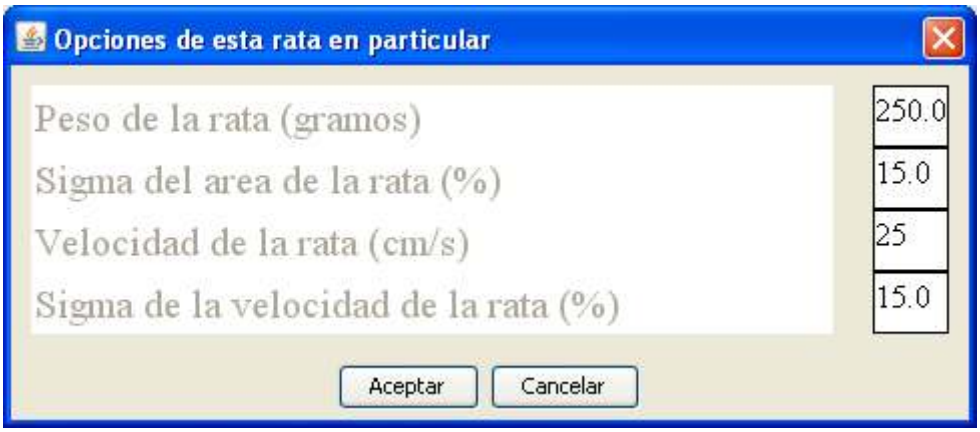
Opciones del animal

En este diálogo podemos elegir entre dos opciones: **Marcar en la rata**, u **Opciones de la rata**.

La opción pinchar en la rata muestra un nuevo diálogo con la imagen del laberinto y el animal, y se nos pide hacer *click* en el animal. La imagen que se muestra en este diálogo es el primer instante del experimento. De esta manera podemos saber la posición de la rata en el primer instante, y aseguramos así la fortaleza del algoritmo de detección.

Diálogo para marcar la rata

Cuando seleccionamos la opción de pinchar en la rata, nos aparece el siguiente diálogo donde insertar las características propias del animal:



Opciones del animal

Antes de realizar el experimento, el animal debe ser pesado para calcular la dosis a administrar al animal. Se decidió que haríamos uso de este hecho para el proyecto. Con el **peso del animal** podemos saber el tamaño del mismo en la imagen, lo cual como se explicará más adelante sirve el procesamiento de imágenes.

El **sigma del área de la rata** es la posible variación del área de la rata en la imagen debido a los movimientos del animal. La cámara está situada encima del laberinto, por ello, el área que se ve en la imagen cambia cuando el animal está levantado, extendido o en torsión.

La **velocidad de la rata** nos sirve para determinar la distancia máxima que puede haber entre el animal en un instante y el animal en el instante siguiente. Esto ayudará a diferenciar el objeto y saber si han ocurrido errores en la detección.

El **sigma de la velocidad de la rata** tiene la misma función que el sigma del área, ya que el animal puede moverse a diferentes velocidades durante el experimento.

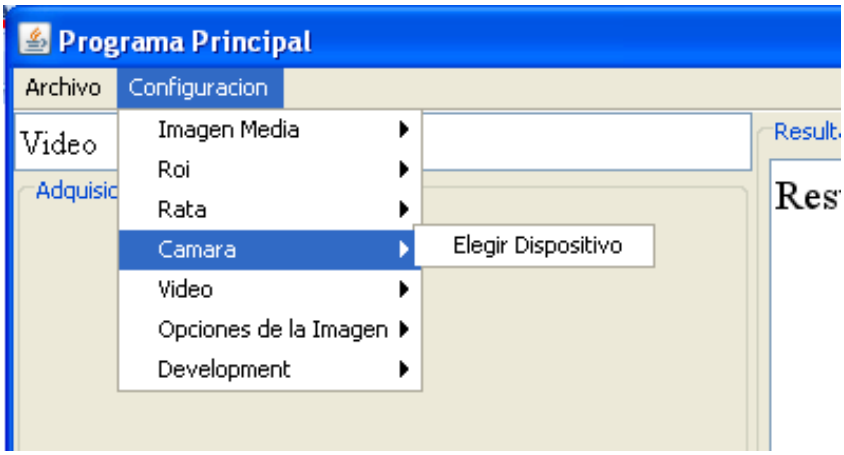
Al introducir el valor de la velocidad del animal en el diálogo anterior, es necesario hacer una operación posterior con el valor introducido. Esto se debe a que la distancia que recorre el animal en pixeles depende no sólo de la máxima velocidad de éste, sino que también depende del número de pixeles que tiene un metro real, además

del *framerate*. En los siguientes apartados se explica cómo se calculan el *framerate* y el número de pixeles por metro. El código que se ejecuta después de introducir el valor es el siguiente:

```
// Si es por segundo, habrá que tener en cuenta el frame rate
// pixelByMetre en m, ratSpeed en cm
StatisticsTools.ratSpeed = ratSpeed*0.01f; // ratSpeed es el valor introducido
StatisticsTools.MOVE_MOVE=StatisticsTools.ratSpeed * StatisticsTools.pixelByMetre * fFrameRate ;
```

Tras aceptar estos valores, se almacenan en la clase *StatisticsTools* para usarlos durante el algoritmo de detección.

CÁMARA

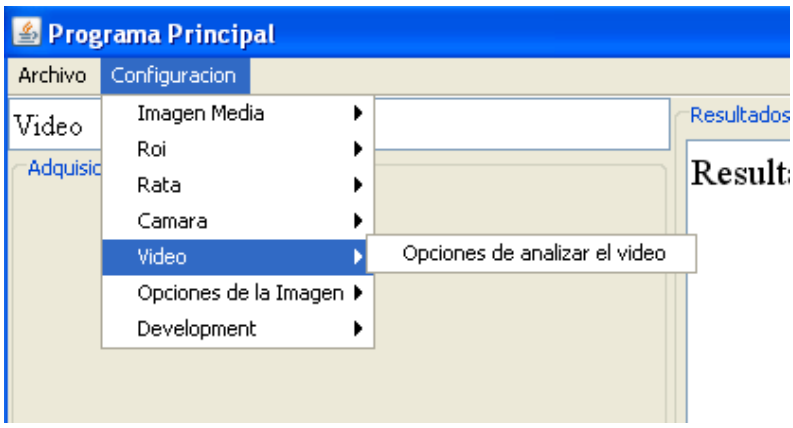


Opciones de la Cámara

En este submenú sólo se nos presenta una opción, **Elegir Dispositivo**, con el cual tendremos acceso al mismo diálogo que obteníamos al intentar abrir la cámara. Como se dijo anteriormente, aquí elegiremos el dispositivo instalado en JMF que queremos utilizar y luego ya podremos usarlo para adquirir las imágenes.

VIDEO

Este submenú contiene una única opción, **opciones de analizar el video**. Haciendo *click* en él iremos a otro diálogo donde elegiremos el tiempo a grabar y el tiempo entre capturas.



Opciones de Video

El diálogo que se genera contiene información sobre el tiempo total de duración del video, si hemos elegido alguno. Si hemos elegido imágenes o cámara no nos dará ningún valor. El diálogo es el siguiente:

Opciones de análisis de video

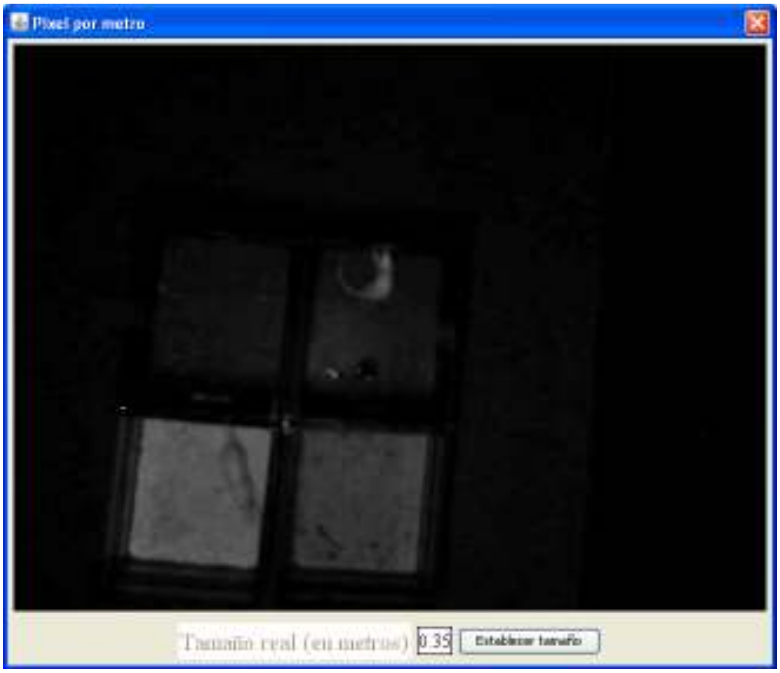
El tiempo entre capturas es lo que antes llamábamos *framerate*. Como hemos dicho se usará para múltiples cosas, entre ellas para determinar la distancia máxima que puede recorrer el animal entre cada toma.

OPCIONES DE LA IMAGEN

Opciones de la imagen

En este submenú, de nuevo, sólo tenemos una opción, el **tamaño de pixel**. Como ya se explicó anteriormente, es necesario conocer cuántos pixeles conforman un metro en la imagen. Como este valor cambia según el tamaño de la imagen (320x240 pixeles, 640x480 pixeles), y de la distancia de la cámara al laberinto, será necesario realizar esta operación cada vez que se coloque la cámara o se mueva el laberinto.

Para realizar esta operación nos ayudaremos del siguiente diálogo y del conocimiento de una distancia exacta en la imagen:



Pixel por metro

Sabiendo la distancia de un elemento en la imagen es fácil calcular los pixeles por metro. En primer lugar señalaremos dos puntos (por ejemplo los dos extremos de un habitáculo), y después introducimos el tamaño real del objeto (en el caso del habitáculo 0.35m).

Después de ello el programa realiza los siguientes pasos. Primero se calcula la distancia en pixeles de los dos puntos y con la distancia real y una sencilla regla de tres obtenemos los pixeles que conforman un metro. El código que realiza esto es el siguiente:

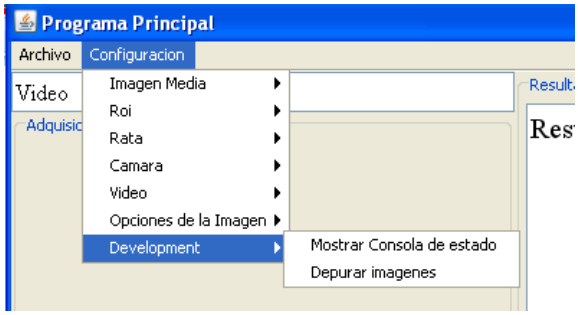
```
// Con los dos puntos calculo su distancia en pixeles
Point2D.Float point = (java.awt.geom.Point2D.Float) vPoints.get(0);
Point2D.Float pointNext = (java.awt.geom.Point2D.Float) vPoints.get(1);

// Distancia de un punto a otro
double pixSize = (double)point.distance(pointNext);

// Regla de tres -> Si pixSize = realSize --> pixByMetre = pixSize / realSize ;
pixelByMetre = pixSize / realSize ;
```

DEVELOPMENT

Para facilitar las labores de depuración y conocer los posibles fallos del programa sin necesidad de compilar las fuentes y usar un programa de depuración externo se ha incluido al programa una **consola** que mostrará

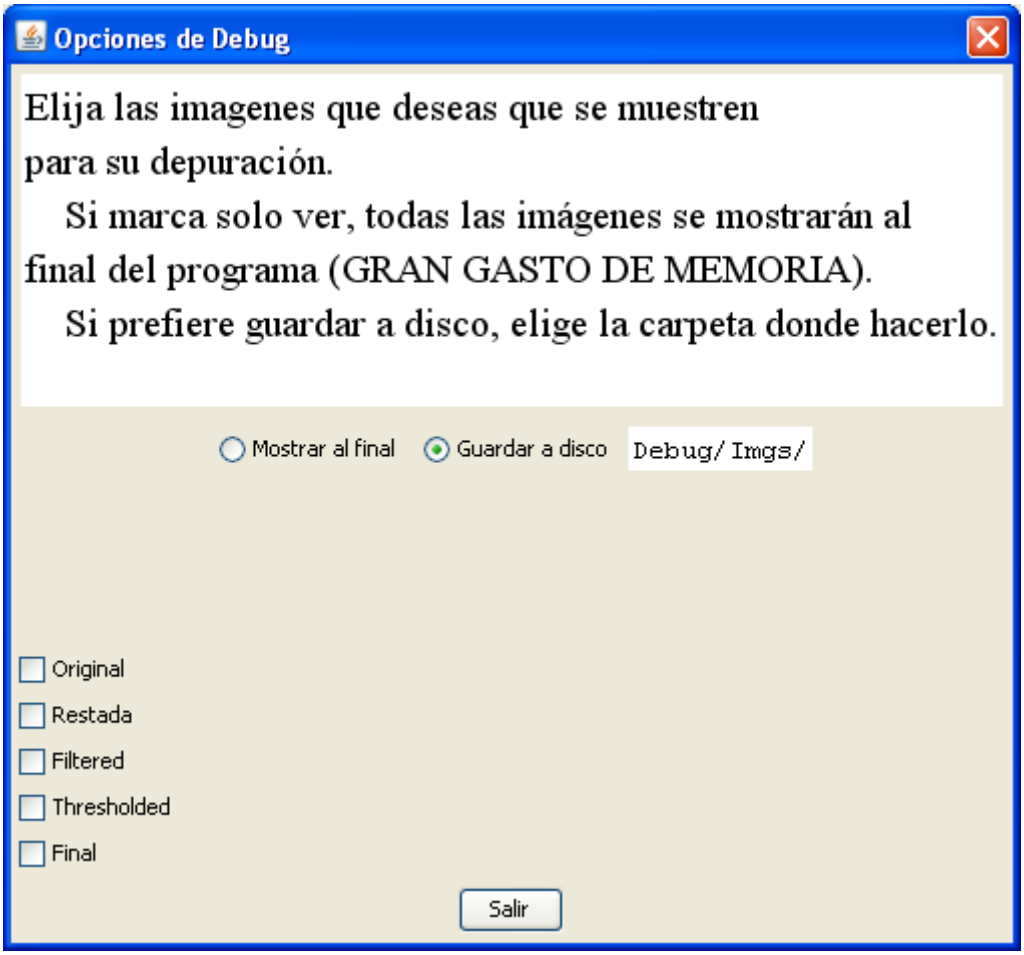


Opciones de depuración

información sobre el proceso de detección y el programa en general.

Esta opción se ha incluido también con el fin de poder almacenar la depuración de un experimento para, en el caso de ocurrir errores, saber por qué han ocurrido.

La consola de estado es sólo una de las opciones de depuración, otra de ellas es la opción **Depurar imágenes**. Cuando la seleccionemos nos aparecerá el siguiente diálogo:



Depurar imágenes

Este diálogo nos permite guardar las imágenes que se han usado en el procesamiento de imágenes para el algoritmo de detección. De esta manera, observando las imágenes del proceso y la consola de depuración se puede determinar con facilidad cual ha sido el problema en la detección si ha habido alguno.

VELOCIDAD DEL ANIMAL EN EL TIEMPO

Finalmente, cuando terminamos el análisis del experimento, nos aparece una nueva ventana donde podemos observar la velocidad del animal a lo largo del tiempo. Para ello haremos uso de la función *Plot* de ImageJ, la cual nos permite generar un gráfico a partir de los valores de un array.

La velocidad del animal se calcula entre cada toma, ya que conocemos la distancia del objeto en el instante anterior y el instante actual, y conocemos también el tiempo entre tomas.

Un ejemplo de gráfico generado es el siguiente:

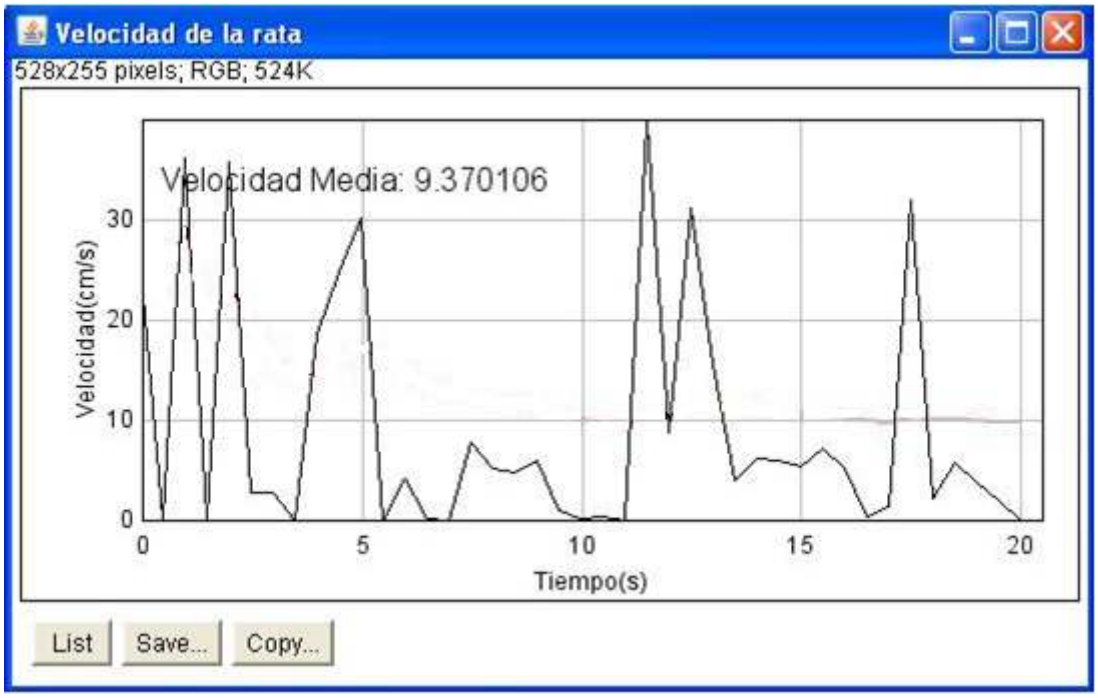


Gráfico de la velocidad del animal

RECIENTES

Como hemos visto a lo largo de todo este apartado, antes de hacer un análisis es necesario inicializar muchas opciones para el correcto funcionamiento. Sin embargo, casi siempre se utilizan las mismas opciones, ya que la cámara no cambia de posición, el animal suele tener siempre el mismo peso y velocidad máxima, y lo único que hace falta cambiar es el video, y pinchar en la posición inicial del animal.

Para solventar esta situación y mejorar la usabilidad del programa se decidió implementar la característica que hemos llamado **recientes**: Cuando se cierra el programa o se termina un análisis, automáticamente se guarda a archivo las opciones del análisis.

Cuando se inicializa el programa, de manera transparente para el usuario se cargan todas estas opciones desde el fichero anterior, para tener listo el programa sin tener que volver a realizar todos los pasos posteriores.

Para implementar esta característica lo primero que se necesitaba era una *clase* que permitiera guardar y cargar opciones del fichero de forma sencilla. Además se necesitaba que estos métodos permitieran trabajar con parámetros de manera independiente, es decir, que se pueda guardar y cargar un solo parámetro del archivo, y no todos a la vez.

La clase implementada contiene los siguientes métodos:

public static void **saveToFile**(String sFile, String msg, int mode)

Guarda el *String* en el archivo pasado. Si el método es de sobre escritura, lo machaca todo encima, si en cambio usamos el método ADD, escribe una traza de control y luego escribe el mensaje

Parámetros:

`sFile` - El archivo donde se quiere guardar

`msg` - El mensaje a guardar

`mode` - Modo: ADD, OVERWRITE

public static void **saveToFile**(String sFile, String sParam, String sValue)

Guarda en el archivo sFile el parámetro sParam con valor sValue. Comprueba si el parámetro ya está en el archivo, si es así cambia su valor, si por el contrario no está guardado en el archivo, crea un nuevo parámetro y guarda su valor. Atención: No hay delimitador en los parámetros, así que es conveniente escribir todos los valores sin espacios para que no haya errores

Parámetros:

`sFile` - Archivo donde se va a guardar

`sParam` - Parámetro a guardar

`sValue` - Valor del parámetro

public static String **readFromFile**(String sFile)

Lee toda la información de un archivo

Parámetros:

`sFile` - El archivo a leer

Devuelve:

La información leída

public static String **readFromFile**(String sFile, String sParam)

Lee un determinado parámetro del archivo de configuración (reciente).

Parámetros:

`sFile` - El archivo a leer

`sParam` - El parámetro a buscar

Devuelve:

El valor en *String* del parámetro o *null* si no hay tal parámetro

Tras tener implementados todos estos métodos, es fácil guardar los parámetros cuando finalizamos un análisis. Guardar un parámetro es tan sencillo como ejecutar la siguiente sentencia:

```
UsingFiles.saveToFile(Recent.sFileRecent, "ratWeight", ""+ratWeight );
```

Como he dicho antes, al iniciar el programa se inicializan todos los parámetros necesarios para realizar el análisis. Para ello se ejecuta la función a continuación. He reducido la opción a sólo a algunos parámetros, ya que se describe perfectamente el proceso con sólo un ejemplo:

```
public void initializeRecent()
{
// CARGAMOS EL PARAMETRO VELOCIDAD DE LA RATA
   if ( UsingFiles.readFromFile(Recent.sFileRecent, "ratSpeed") != null)
      StatisticsTools.ratSpeed = Float.valueOf( UsingFiles.readFromFile(
         Recent.sFileRecent, "ratSpeed") );
// CARGAMOS EL NOMBRE DEL ARCHIVO DEL ROI
   if (UsingFiles.readFromFile(Recent.sFileRecent, "roiFile") != null)
   {
      String sRoiRecent = UsingFiles.readFromFile(Recent.sFileRecent, "roiFile");
      File file = new File(sRoiRecent);
      if ( file.exists())
         GetRoiTools.readRoiFile(new File(sRoiRecent));
   }
// SI SE HA CARGADO EL ARCHIVO CARGAMOS EL ROI
   if (UsingFiles.readFromFile(Recent.sFileRecent, "averageImageFile") != null)
   {
      String sAverageFile = UsingFiles.readFromFile(Recent.sFileRecent, "averageImageFile");

      Opener opener = new Opener();
      ImagePlus img = opener.openImage(sAverageFile);
      if (img != null)
      {
         analyze.ipAverage = img.getProcessor().duplicate();
         // Muestro la imagen
         menuItemShowAverageActionPerformed();
      }
   }
}
```

ADQUISICIÓN DE LA FUENTE

El primer paso necesario para nuestro programa es adquirir la fuente. Como ya hemos dicho antes, inicialmente disponemos de una cámara, pero el conversor analógico / digital es usado por un programa propietario, por lo que no podemos usar esta fuente. Por tanto, vamos a usar una cámara web convencional, que tiene un coste mucho menor que las profesionales, a costa de proporcionar una imagen con una baja relación SNR (la imagen adquirida tiene mucho ruido).

Para trabajar con cámaras web, utilizaremos JMF (Java Media Framework), un conjunto de herramientas proporcionadas por Java que permiten trabajar con formatos multimedia, tales como audio, video, tarjetas capturadoras…

JMF

El API de *Java Media Framework* especifica una arquitectura simple y unificada para sincronizar y controlar audio, video y otros datos basados en tiempo en aplicaciones Java y *applets* (componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web).

El API de JMF explica el comportamiento de la reproducción, captura y transcodificación de fuentes multimedia y fue desarrollado por *Sun Microsystems* e *IBM*.

El programa puede trabajar con diversas fuentes. Las imágenes son tratadas con el API de ImageJ, mientras que las fuentes de video y cámara se obtienen mediante JMF. Este API trabaja mediante reproductores (*Players*). Para crear un reproductor ejecutamos utilizaremos la clase *Player* y *Manager*:

```
/**
* Obtiene el reproductor a través de un archivo de video.
*
* @param file El archivo de video a abrir
* @return El reproductor
*/
static Player getPlayer(File file)
{
   playerFile = file;
   // Seteo el manager para poder usar SWING
   Manager.setHint(Manager.LIGHTWEIGHT_RENDERER, new Boolean(true) );

   // Creo el url del File
   URL urlVideoFile = null;
   try {
      urlVideoFile = new URL("file:" + file.getPath());
   } catch (MalformedURLException ex) {
      ex.printStackTrace();
   }

   Player player = null;
   try {
      player = Manager.createRealizedPlayer( urlVideoFile );
   } catch (NoPlayerException ex) {  ex.printStackTrace();
   } catch (CannotRealizeException ex) { ex.printStackTrace();
   } catch (IOException ex) { ex.printStackTrace(); }

   player.prefetch();

   try {
      Thread.sleep(1000);
   } catch (InterruptedException ex) { ex.printStackTrace(); }

   return player;
}
```

Un *Manager* es un punto de acceso para obtener sistemas que dependen del tiempo como reproductores. Lo primero que hay que hacer con el *Manager* es definir el tipo de componente que vamos a crear:

```
Manager.setHint(Manager.LIGHTWEIGHT_RENDERER, new Boolean(true) );
```

Con la sentencia anterior especificamos que el *Manager* va a ser un componente de peso ligero, que es el tipo de componente que necesita SWING (el API de la interfaz gráfica) para poder insertarlo en la interfaz gráfica.

Tras definir el tipo de componente que va a devolver el Manager, procedemos a crear un reproductor (un *Player*) con el mismo Manager:

```
player = Manager.createRealizedPlayer( urlVideoFile );
```

Esta función necesita una *URL*, la dirección física del archivo que queremos reproducir, y nos devuelve el reproductor creado o error. *Realized* significa que el reproductor está lanzado, que contiene los controles para poder visualizarlo y que está listo para visualizar o trabajar con él.

La función anterior sirve para crear un reproductor a partir de una *URL*, pero cuando queremos crear un *player* a partir de un dispositivo es necesario realizar algunos pasos previos para obtener un *MediaLocator*. El *MediaLocator* es un identificador de todos los dispositivos que se encuentran instalados en JMF. Éste identificador se obtiene cuando se selecciona en la pantalla de elección de cámara:

Pantalla de elección de dispositivo

El código necesario para obtener el *MediaLocator* y crear un *RealizedPlayer* a partir de él es el siguiente:

```
// Establezco el formato del player
 Format[] cfmts = dev.getFormats();
// Elijo YUV o RGB
Format formatSelected = null;
 if( sFormat.substring(0,3).equals("rgb") )
{
     RGBFormat fmt = (RGBFormat)cfmts[ intFormatSelected ];
    formatSelected = fmt;
}
if( sFormat.substring(0,3).equals("yuv") )
{
    YUVFormat yuvF = (YUVFormat)cfmts[ intFormatSelected ];
    formatSelected = yuvF;
}
// si no elegí ninguno, cojo el primero que haya
if (formatSelected == null)
    formatSelected = cfmts[0];

//obtengo el locator del dispositivo
MediaLocator loc = dev.getLocator();

// Para poder añadir un player a swing necesitor hacer esto
Manager.setHint(Manager.LIGHTWEIGHT_RENDERER, true);

// Intento crear el reproductor
try {
    player = Manager.createRealizedPlayer(loc);
} catch (Exception ex) {
    ex.printStackTrace();
}

// Establezco el formato del player
FormatControl formatControl = (FormatControl)player.getControl ( "javax.media.control.FormatControl"
        formatControl.setFormat( cfmts[ intFormatSelected ] );

player.start();
```

Después de ejecutar estas instrucciones, y si no ha habido errores obtenernos un reproductor asociado a la cámara instalada en JMF.

OBTENER IMAGEN DE GRISES DEL REPRODUCTOR

Lo siguiente que es necesario hacer con el *Player* es obtener una imagen en blanco y negro (nuestro algoritmo trabaja con imágenes en escala de grises de 8 bits). Para ello usamos la siguiente función:

```
/**
* Obtener una imagen de grises del reproductor
*
* @param player    El reproductor de donde coger la imagen
* @return          La imagen de grises
*/
static ImageProcessor getGrayImageFromPlayer(Player player)
{
   // capturamos la imagen en un buffer
   FrameGrabbingControl fgc = (FrameGrabbingControl)
   player.getControl("javax.media.control.FrameGrabbingControl");
   Buffer buf = fgc.grabFrame();

   // creamos la imagen awt y la devolvemos
   BufferToImage btoi = new BufferToImage((VideoFormat)buf.getFormat());
   Image img = btoi.createImage(buf);
   ImagePlus imgPlus = new ImagePlus("",img);

   ImageProcessor ip = imgPlus.getProcessor();

   ip = ip.convertToByte(false);
   ip.smooth();

   return ip;
}
```

Pero esto no es lo único a tener en cuenta con la fuente, sobre todo en la cámara. En la interfaz gráfica había un apartado donde definíamos el *FrameRate*:

Opciones de análisis de video

El reproductor no conoce cuál es el *FrameRate* que tiene aplicar, es decir, no sabe cada cuanto tiempo tiene que tomar una imagen. Por ello tiene que hacerse manualmente a la hora de realizar el análisis, siguiendo los siguientes pasos:

```
// Recojo el controller para poder elegir frame
FramePositioningControl fpc = (FramePositioningControl) player.
   getControl("javax.media.control.FramePositioningControl");

Time initialTime = player.getMediaTime();// Obtengo el tiempo actual
Time finalTime = new Time(initialTime.getSeconds() + fAnalyzeTime ); // Obtengo el tiempo final

// Hago un seguimiento del numero de cortes
int sliceNumber = 0;
// Desde initialTime hasta finalTime en intervalos de FrameRate
for (double second=initialTime.getSeconds(); second <= finalTime.getSeconds()-fFrameRate;
    second += fFrameRate)
{
   // Recojo el frame que me marque second
   System.out.println("second -> " + second);
   fpc.seek( fpc.mapTimeToFrame(new Time(second)) ); // Muevo el player
   ImageProcessor ip = PlayerTools.getGrayImageFromPlayer(player); // Obtengo la image
         // REALIZO EL ANÁLISIS DE LA IMAGEN
}
```

Análisis de un video

En el caso de un video es sencillo mover el reproductor a un tiempo determinado, hacer una captura, realizar el análisis y mover el video al siguiente instante.

Sin embargo, realizar esta misma operación en tiempo real no es igual de sencillo. En primer lugar no sabemos cuánto va a tardar el algoritmo de detección, por lo que no podemos saber si el *FrameRate* elegido se está cumpliendo. El tiempo que tarda en obtener la imagen y procesarla debe ser menor que el *FrameRate*.

Cuando el tiempo de operación es mayor, evidencia la necesidad de un tiempo menor de proceso, es decir de un PC de mayores prestaciones. Cuando el tiempo en obtener la imagen y procesarla es menor que el *FrameRate* es necesario parar la ejecución del análisis el tiempo sobrante, para no incurrir en problemas de cálculo de velocidades, de determinación de distancia del objeto entre imágenes, etc.

El algoritmo que realiza las operaciones anteriores es el siguiente:

```
double initialTime = System.currentTimeMillis();
double finalTime = initialTime + fAnalyzeTime*1000;

// Desde el tiempo actual hasta el tiempo que se quiere analizar, obtengo
// la imagen del player, y la analizo -> analyze.analyzeFrame()

// Realizo un seguimiento el tiempo y el numero de cortes
double actualTime = initialTime;
int sliceNumber = 0;
// Hasta que actualTime supere a finalTime
while (actualTime < finalTime) {
   System.out.println("\n\nSliceNumber-> " + sliceNumber + "  Tiempo: " + actualTime);

   // Recojo el frame en ese momento
   ImageProcessor ip = PlayerTools.getGrayImageFromPlayer(player); // Obtengo la imagen

         // ANÁLISIS DE IMAGEN

   // Si he tardado menos de lo que marca fFrameRate, entonces tengo que esperar
   double processSeconds = (System.currentTimeMillis() - actualTime)/1000 ;
   if ( processSeconds < fFrameRate )
   {
```

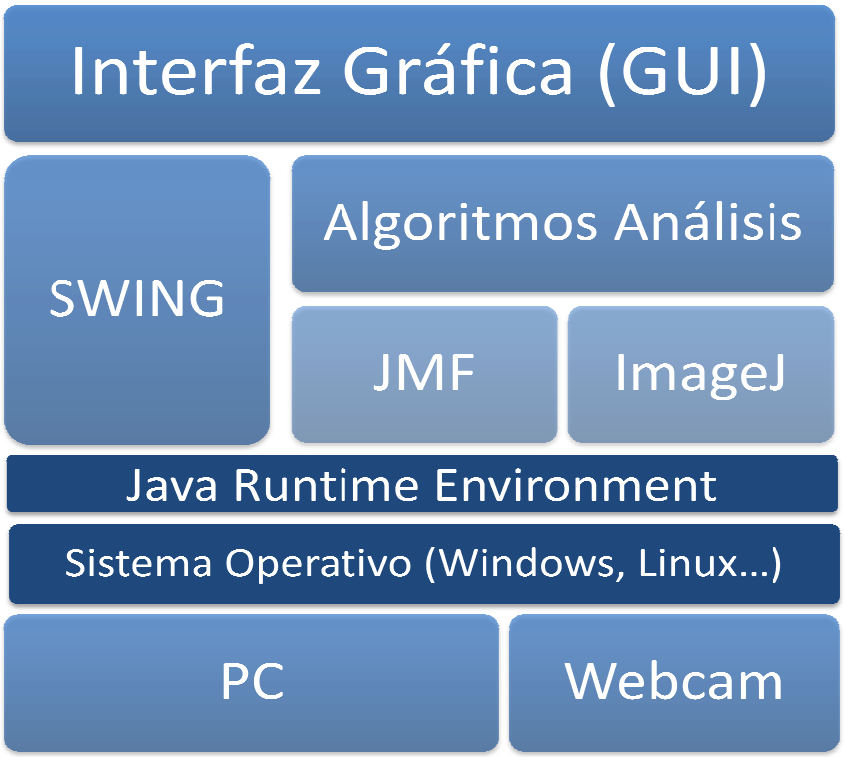
```
        double toSleep = (fFrameRate - processSeconds);
        try {
            Thread.sleep( (long)(toSleep*1000) );
        } catch (InterruptedException ex) { ex.printStackTrace(); }
    }
    else // Si he tardado más tiempo informo del hecho
        System.out.println( "Tiempo excedido en el proceso: + (processSeconds-fFrameRate) );

    // Actualizo el tiempo actual
    actualTime = System.currentTimeMillis();
    System.out.println("actualTime -> " + actualTime + "  finalTime -> " + finalTime);
}
```

Análisis de cámara

ESTRUCTURA DEL PROGRAMA

En este apartado vamos a analizar la estructura del programa, qué llamadas se hacen al API de Java, a JMF para adquirir imágenes de la cámara web y al algoritmo propio. Para una mejor representación se presentará una pirámide donde se puede observar esta estructura:



Pila de Aplicaciones utilizadas

Como se puede observar, el programa implementado hace uso de diversas tecnologías, todas ellas de libre uso y basadas en Java. Swing es un *framework* para crear interfaces. JMF (*Java Media Framework*) es otro paquete que permite la interactividad con fuentes multimedia, y es el encargado de recoger las imágenes de la webcam. ImageJ como hemos dicho anteriormente es el programa de gestión de las imágenes y videos, además del encargado de aplicar algunas funciones ya implementadas en las imágenes.

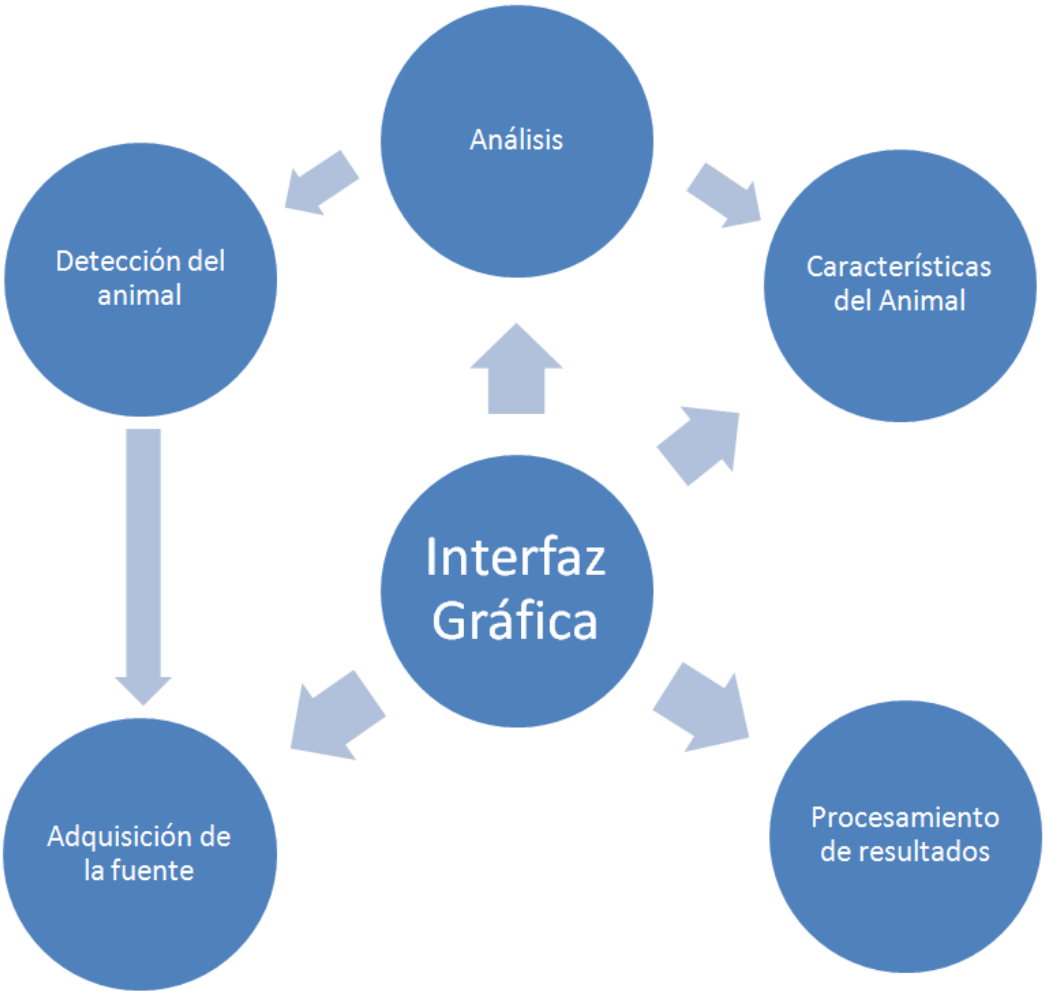
El programa se ejecuta sobre el *Java Runtime Environment*, que se encuentra disponible para múltiples plataformas, y nos permite ejecutarlo sobre multitud de sistemas operativos.

Y por último el sistema físico: el PC y la cámara.

Ya hemos dicho anteriormente que el programa es modular, vamos a explicar ahora este concepto. Modular significa que cada parte del programa está escrita de manera independiente, con un esqueleto que permite el paso de parámetros y la devolución de resultados, pero sin modificar ninguno de los otros módulos.

De esa manera, el módulo principal, que es la interfaz gráfica, va llamando cuando el usuario lo requiere a cada uno de los otros módulos. Es evidente que algunos módulos necesitarán de otros para poder ejecutarse, por lo que el programa nos advertirá cuando se intente ejecutar, para que cumplamos los requerimientos.

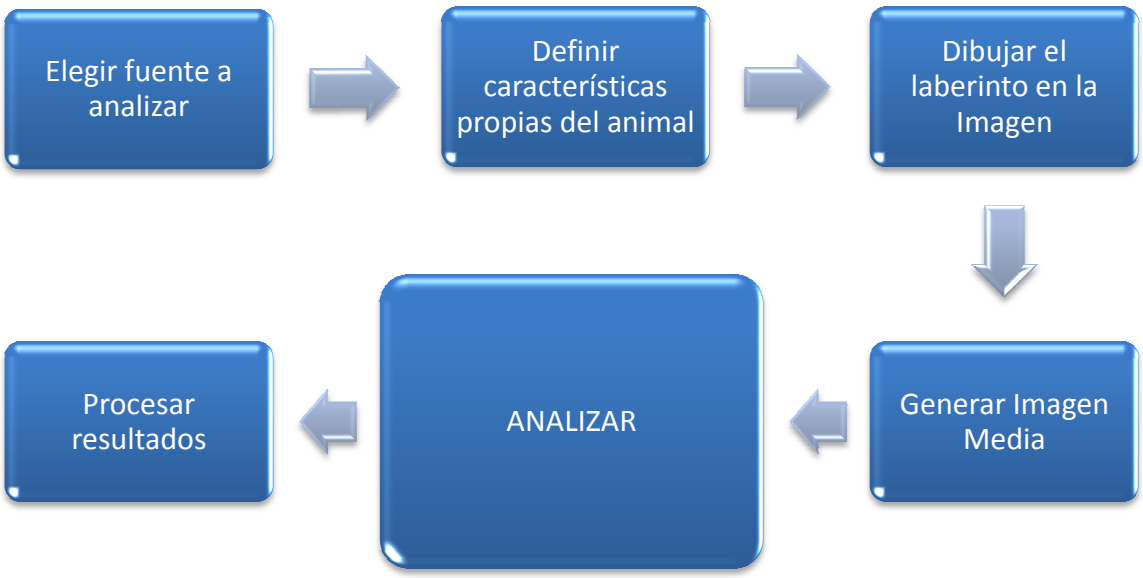
Estos módulos se pueden ver en el siguiente gráfico. En él se introducen algunas funcionalidades que serán largamente explicadas en sucesivos apartados.



Estructura de la implementación

USABILIDAD

Es importante también explicar cuál es el funcionamiento normal del programa, el cual se ve claramente en el siguiente gráfico de flujo:



Flujo de programa

El primer paso del programa es elegir la **fuente a analizar**, aquí podemos decidir entre imágenes almacenadas, un vídeo guardado, o imágenes en tiempo real de la cámara. Una vez elegido se muestra en la interfaz la primera imagen, el primer instante del video o la imagen de la cámara. En el caso de elegir un video o tiempo real hay que especificar que *framerate* se quiere usar, es decir, la frecuencia con la que se genera una imagen fija.

Lo siguiente es **definir las características propias del animal**. Para ello introduciremos en un diálogo de la interfaz características como velocidad máxima y peso. Para el caso de una rata es 300gr y 25cm/s.

A continuación se procede a **dibujar el laberinto**. En este paso definimos el número de habitáculos y se representan en una imagen. Esta imagen se imprime en un nuevo diálogo donde el usuario selecciona cada esquina del laberinto definiendo el polígono. Una vez terminado, y mediante un *callback* a la interfaz principal se guarda el laberinto para usarlo durante el análisis.

La **generación de la imagen media** es un paso a medio camino entre el análisis y la generación de las características propias del experimento. Es un paso muy importante para realizar un buen análisis, y por ello dedicaré más adelante una larga explicación. Como introducción explicar que la imagen media hace uso tanto del laberinto como de la fuente.

Cuando se terminan estos pasos previos se pasa al **análisis**. En esta parte se hace uso de JMF y de un algoritmo propio para detectar la posición del animal en la imagen. Para cada *slice* (así se define cada imagen) se almacena un objeto animal, que contiene las coordenadas del animal en la imagen, y otro parámetro interesante: si ha habido problemas a la hora de detectar la posición. Más adelante se explicará cómo se detecta la posición y se determina si ocurrieron problemas en la detección.

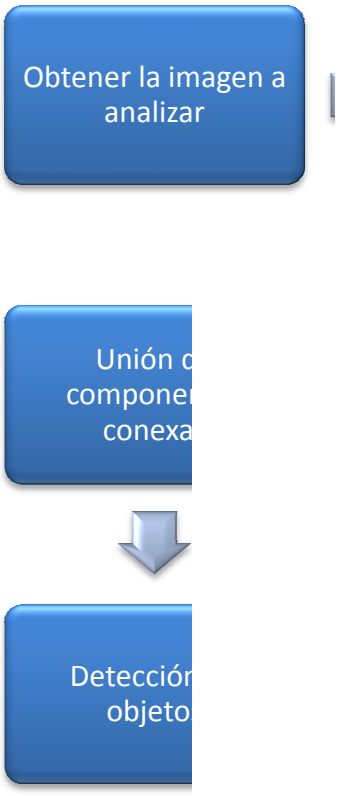
Finalmente se **procesan los resultados**, donde se analiza la trayectoria y se pueden re-analizar los *slices* conflictivos. Este método realiza un análisis inverso en el tiempo, para corregir las detecciones incorrectas.

Más adelante veremos en profundidad cada uno de estos procesos, sobre todo la parte de **análisis**, donde definiremos cada uno de los pasos dados en el procesamiento de las imágenes.

ALGORITMO DE ANÁL

Tras determinar toc
programa lanza un hilo pa
un instante de tiempo).

La estructura del
operaciones que serán ex
algoritmos de detección (u
cambia la forma de obtener



Desde el principio
pero no hemos dado suficie

Un ejemplo de est
hecho de que para cada
algoritmo.

Este algoritmo no diferencia entre la manera de adquirir las imágenes. La única información que el algoritmo necesita para la adquisición de las imágenes es el tiempo total del experimento, el intervalo de tiempo entre cada una de las imágenes, y por supuesto la imagen tomada cada cierto tiempo.

De esta manera es más fácil detectar errores, y llevar un control del proceso más exhaustivo, ya que siempre tenemos las mismas trazas, las mismas sentencias ejecutadas, etc.

El error es independiente de donde se obtuvo la fuente, si el algoritmo de detección ha fallado se debe a un motivo externo a la fuente, y gracias a esto se puede solucionar en un sitio determinado y esa corrección servirá para cualquier manera de adquirir las imágenes.

ALGORITMO DE FUNCIONAMIENTO

En la página siguiente se encuentra el algoritmo implementado, pero en vez de pegar el código usado, vamos a usar un gráfico de flujo, ya que es mucho más sencillo de entender el planteamiento, y no nos perderemos en vicisitudes de nombres de clases, de métodos y sobre todo nos distanciaremos del lenguaje elegido, para valorar la viabilidad del algoritmo

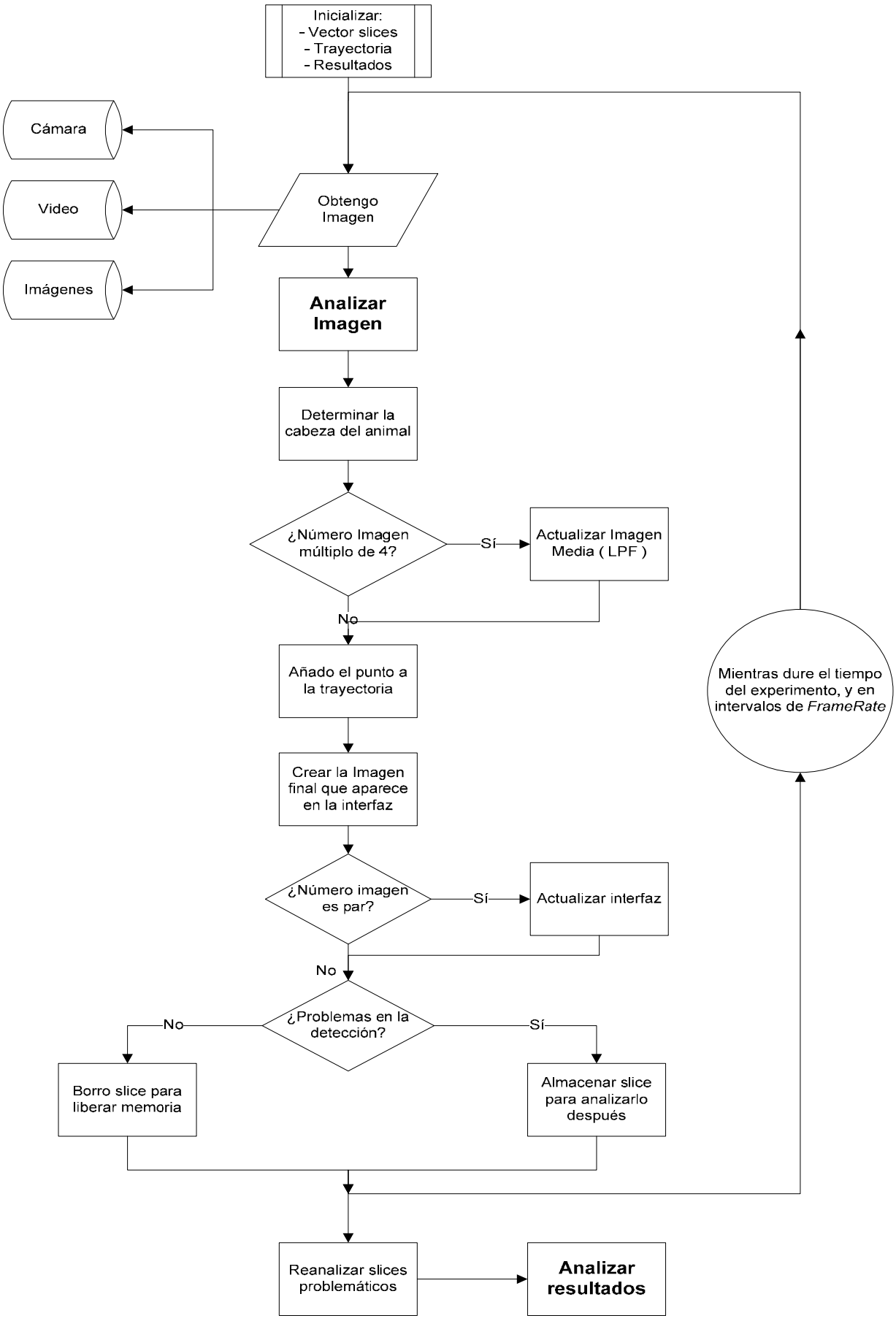


Diagrama de flujo del funcionamiento del programa

El gráfico de flujo anterior presenta una parte del algoritmo de detección del animal a muy alto nivel, en el cual aún no hemos entrado en conceptos como por ejemplo procesar la imagen, detectar objetos, etc.

Sin embargo es muy útil para hacer una primera aproximación a la detección, pero incluyendo a su vez procesos en los cuales interactuamos con la interfaz gráfica para mostrar el animal, o el posterior análisis de las detecciones problemáticas.

En primer lugar el algoritmo inicializa los vectores donde se almacenan la trayectoria, los resultados y los *slices*. La trayectoria es un vector de tipo FIFO (primero en entrar, primero en salir) donde se almacenan las últimas 10 posiciones del animal, y nos sirve para representarlo en la imagen de la interfaz. El vector de resultados contiene los resultados que se muestran en la ventana principal de la interfaz. Y por último, el vector *slice* contiene un *objeto animal* por cada imagen tomada.

Para este objeto se ha determinado crear una clase que contenga las siguientes variables para cada uno de los objetos:

- Imagen original
- Imagen binarizada durante el proceso de *Analizar Imagen*
- Vector de puntos representativo del animal. En este vector se almacenan todos los puntos que durante el algoritmo de detección se han determinado como pertenecientes al animal.
- Vector con los habitáculos afectados: Se explicará más adelante qué significa esto de los habitáculos afectados (o ROIs afectados).
- Posición determinada del animal.
- Posición de la cabeza del animal: Más adelante se verá otro gráfico de flujo donde se indica la forma de obtener esta posición.
- Posición y ROIs afectados en el instante anterior

Variables de la clase *slice* (objeto animal)

La clase del objeto animal no sólo contiene las variables mencionadas, sino que también implementa los métodos con los que determinar la posición del animal, su cabeza, los ROIs afectados, etc.

Por tanto, el algoritmo de detección de la posición primero crea un objeto animal, el cual se inicializa con la imagen original (la cual se ha obtenido de la cámara, de un video o de un archivo imagen). Posteriormente se procesa esta imagen, y se almacena el resultado de este proceso en el objeto animal.

Una vez hecho esto se llama a cada uno de los métodos implementados en el objeto animal que determinan la posición del mismo, los ROIs afectados, la cabeza de éste, etc.

Después de hacer esto se ejecutan algunas rutinas como actualizar la imagen media (proceso que se explicará más adelante), se añade el punto determinado a la trayectoria del animal, y se genera la imagen que se mostrará en la interfaz.

Esta imagen se genera desde cero en negro, y sobre ella se dibujan las líneas que conforman cada uno de los habitáculos (o ROIs), el vector de puntos del animal y la trayectoria en las últimas quince imágenes. También se actualizan los resultados que se presentan al usuario en tiempo real.

Una vez preparado esto se actualiza la interfaz (se *refresca*) cada dos imágenes. Esto se ha decidido implementar de esa manera para mejorar la efectividad del método de detección en tiempo real (con la cámara), ya que no perdemos tanto tiempo de proceso si actualizamos la interfaz sólo cada dos imágenes tomadas.

Tras esto se determina si ha ocurrido algún error en la detección del animal mediante un método implementado en la clase del objeto animal, el cual explicaremos más adelante en profundidad. Si ha habido problemas se almacena este objeto para efectuar otro análisis más adelante. Pero no se guarda sólo este objeto, sino que se guardan también el objeto perteneciente al instante anterior y posterior del problemático.

Guardar los instantes anterior y posterior nos servirá para tener información del momento en el que se volvió a detectar al animal, y poder hacer un análisis temporalmente inverso, y comprobar si la trayectoria que ha seguido el animal es posible según unos patrones establecidos (también se explicará en profundidad más adelante).

Si no han ocurrido problemas se llama al método implementado que libera memoria del objeto animal. Este método elimina las imágenes del objeto, además de las variables que ya no se necesitan, como por ejemplo información sobre el instante anterior.

Después de esto se repite el ciclo recogiendo otra imagen y realizando los mismos pasos descritos anteriormente.

Finalmente, cuando el tiempo del experimento ha concluido se llama a la función para re-analizar las imágenes problemáticas, y si se requiere, se realiza un análisis de los resultados. Más adelante se explicará en mayor profundidad que se puede hacer con los resultados y para qué es necesario su análisis.

ANÁLISIS DE LA IMAGEN

En el siguiente gráfico vamos a explicar el funcionamiento del algoritmo de análisis de imagen. Igual que se ha hecho con el gráfico anterior, primero vamos a insertarlo en la imagen y posteriormente se describirán los principales pasos que se pueden observar.

El método de análisis de imagen es llamado desde el algoritmo anterior y se le pasan tres parámetros: la imagen original, la imagen media y el número de imagen tomada.

El primer paso que realiza el algoritmo consiste en crear un nuevo objeto animal con el número, la imagen original, y guarda en una variable las dimensiones de la imagen ya que serán usadas en el procesado de imágenes.

Como ya se ha comentado, el análisis del animal hace uso de información de la posición del mismo en el instante anterior. Por ello, el algoritmo comprueba si es la primera imagen. En caso afirmativo, se usa la información obtenida cuando marcábamos el animal en el instante inicial. Se genera un polígono alrededor del punto marcado y se establece la posición anterior a ese punto y al polígono generado. Generar un polígono nos sirve para calcular los ROIs afectados (método que se explicará más adelante), ya que veremos que se necesita información de la longitud del animal.

En el caso de que no sea la primera imagen, se recoge la información del instante anterior: la posición y el polígono. De nuevo obtenemos los habitáculos o ROIs afectados, y procedemos a binarizar la imagen.

Binarizar la imagen consiste en procesar la imagen de tal manera que el resultado sea la misma imagen pero con dos únicos valores (en lugar de la escala de grises con la que partíamos). Uno de estos valores será el objeto buscado, y el otro valor se asignará al denominado *fondo de la imagen*. Es decir, el resultado será una imagen negra con el objeto en blanco (o viceversa).

Debido a que el método de binarización realiza un procesado de imagen complejo y que necesita ser explicado en profundidad dejaremos para más adelante esta explicación y nos centraremos ahora en el funcionamiento de este algoritmo.

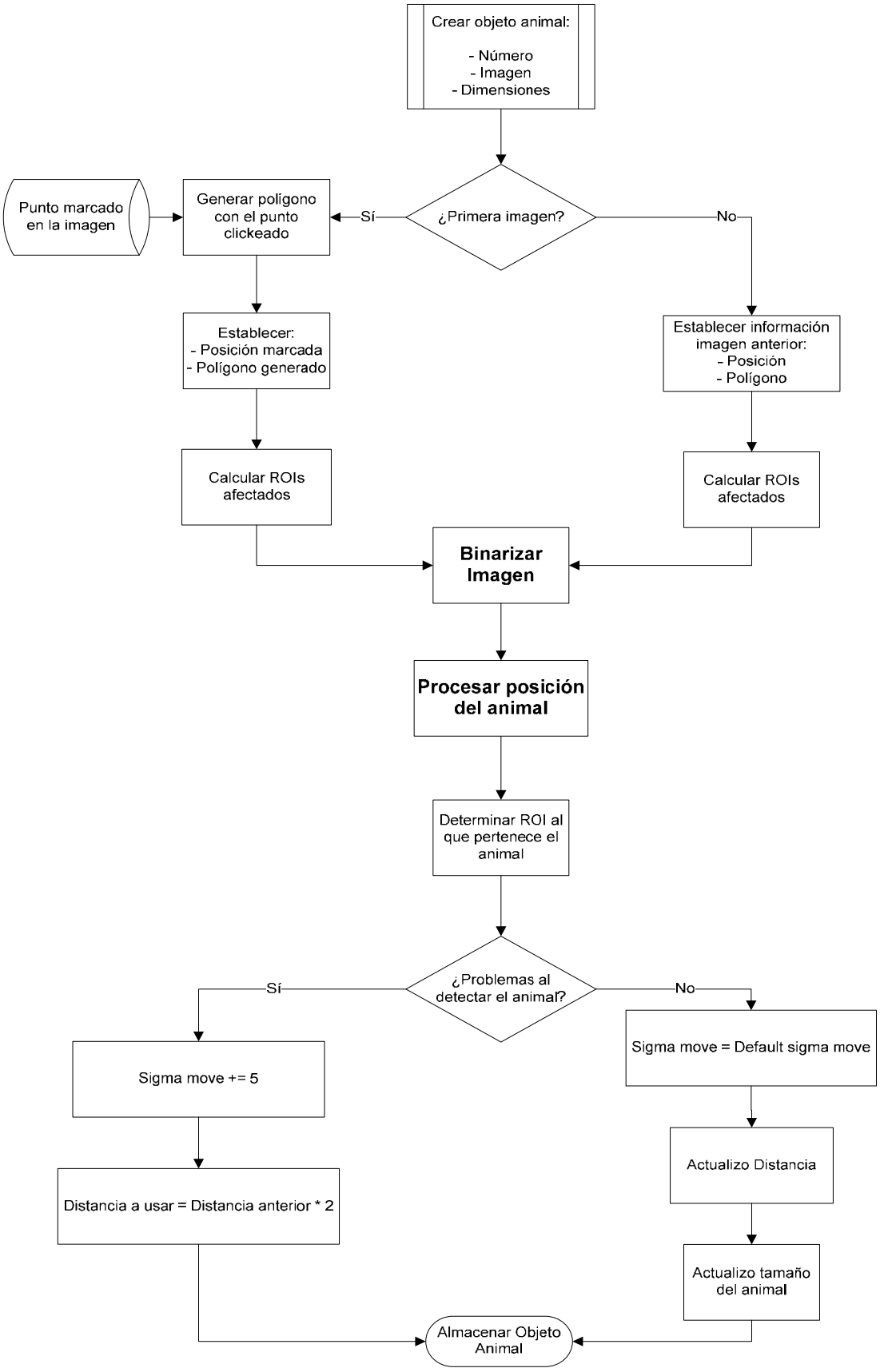


Diagrama de flujo del algoritmo detección del animal

Con la imagen binarizada procedemos a detectar la posición del animal. De nuevo este algoritmo es más complejo y lo explicaremos más adelante, y para el proceso en el que nos encontramos lo único que necesitamos saber es que el método se ejecuta sobre el objeto animal (la clase que hemos estado comentando), la cual contiene la imagen binarizada, y que almacena en la variable *vector de puntos del animal* el objeto detectado y elegido de esta imagen binaria.

Tras esto se calcula cuál es el habitáculo al que pertenece el animal y se almacena de la misma forma en una variable del objeto de la clase animal.

El siguiente paso que realiza el algoritmo consiste en comprobar si ha habido algún problema detectando al animal. Para ello tenemos que explicar en profundidad como se detectan los objetos en la imagen binarizada y como se eligen si se detectan más de uno.

Pero para seguir un hilo lógico de explicación primero es necesario explicar cómo se binariza la imagen, por lo cual reservaremos la explicación de la última parte del grafo en la que se hablan de conceptos como *sigma move*, que es uno de los parámetros utilizado en la detección de objetos.

Por tanto, para concluir la explicación de este algoritmo, el último paso que se realiza es el de almacenar el objeto animal en el vector *slices*.

Recapitulando, el algoritmo principal recoge las imágenes cada cierto tiempo durante la duración del experimento, llama al método de análisis de la imagen que se encarga de binarizar la imagen, procesar la posición del animal y guardar un objeto animal en el vector *slices*. Después se efectúan unas rutinas de actualización de la interfaz y de presentación de los resultados que no afecta a la detección del animal. Por último se vuelven a analizar las imágenes problemáticas y se concluye la presentación en tiempo real de los resultados con el tiempo de estancia en cada habitáculo y el porcentaje sobre el total del tiempo del experimento.

BINARIZACIÓN

La binarización de una imagen consiste en un proceso de reducción de la información de la misma, en la que sólo persisten dos valores: verdadero y falso. En una imagen digital, estos valores, verdadero y falso, pueden representarse por los valores 0 y 1, ó, más frecuentemente, por los colores negro (valor de gris 0) y blanco (valor de gris 255).

En el proceso y análisis de imagen, la binarización se emplea para separar las regiones u objetos de interés en una imagen del resto. Las imágenes binarias se usan en operaciones booleanas o lógicas para identificar individualmente objetos de interés o para crear máscaras sobre regiones.

En muchos casos, una imagen binaria es el resultado de una segmentación por niveles de gris o de una segmentación por selección de un rango de color determinado. En otros casos, una imagen binaria es simplemente el resultado de una selección interactiva de regiones de interés, las cuales se utilizarán como mascaras de comparación o referencia.

La segmentación de una imagen consiste en la división o separación de una imagen en regiones de atributos similares. Es decir, la segmentación subdivide una imagen en sus partes constituyentes u objetos. El grado de subdivisiones depende del problema a resolver, por lo que la segmentación deberá detenerse cuando los objetos de interés de nuestro estudio hayan sido aislados.

Los algoritmos para segmentar imágenes monocromáticas generalmente se basan en una de las dos propiedades básicas de los valores del nivel de gris: discontinuidad y similitud. En la primera categoría, la aproximación particiona la imagen basándose en cambios abruptos en el nivel de gris. Las principales áreas de interés en esta categoría son de detección de puntos aislados, de líneas y de bordes en una imagen. La segunda categoría se basa en segmentar por niveles de umbral, en regiones de crecimiento y en corte y pegado de regiones.

Nuestro algoritmo hará uso de la segmentación por umbral, también denominada umbralización.

DEFINICIÓN DE UMBRALIZACIÓN

Al aplicar un umbral determinado a la imagen en escala de grises, ésta quedará binarizada, etiquetando a cero los píxeles que correspondan al fondo, y con uno aquellos que se consideren objetos.

Por ejemplo, si los objetos son claros respecto del fondo, como en nuestro caso, se aplicará:

$$Pixel\ nueva\ imagen(x,y) = \begin{cases} 1 \Leftrightarrow Pixel\ imagen(x,y) \geq Umbral \\ 0 \Leftrightarrow Pixel\ imagen(x,y) < Umbral \end{cases}$$

Es por tanto necesario e imprescindible el uso de un valor umbral que se usará al aplicar la umbralización. El cálculo y determinación de este umbral es dinámico, varía para cada imagen, por tanto es una operación que habrá que llevar a cabo para cada instante de tiempo en el que se quiera determinar la posición del animal.

Hay muchas técnicas y métodos para determinar el valor umbral con el que binarizar la imagen. Para cada tipo de imagen y según lo que se quiera segmentar de la misma existen diferentes métodos.

Aunque, tal como se puede observar en el gráfico siguiente donde se explica el algoritmo de binarización de la imagen, el método usa los algoritmos de *Otsu* y otro que denominaremos *Fuzzy*, en un principio el cálculo del umbral se llevó a cabo de una forma más manual, como se describirá más adelante.

La umbralización es por tanto el proceso que tenemos que aplicar a la imagen para destacar el objeto del fondo. Sin embargo, antes de aplicar esta segmentación es conveniente aplicar una serie de filtros para reducir el ruido de la señal. También se aplican algunos procesos para destacar inicialmente el objeto del fondo, como es el uso de la imagen media.

Todos estos filtros y procesos es lo que podemos llamar procesado digital de la señal y nos servirá para robustecer el algoritmo de detección, como se verá a continuación.

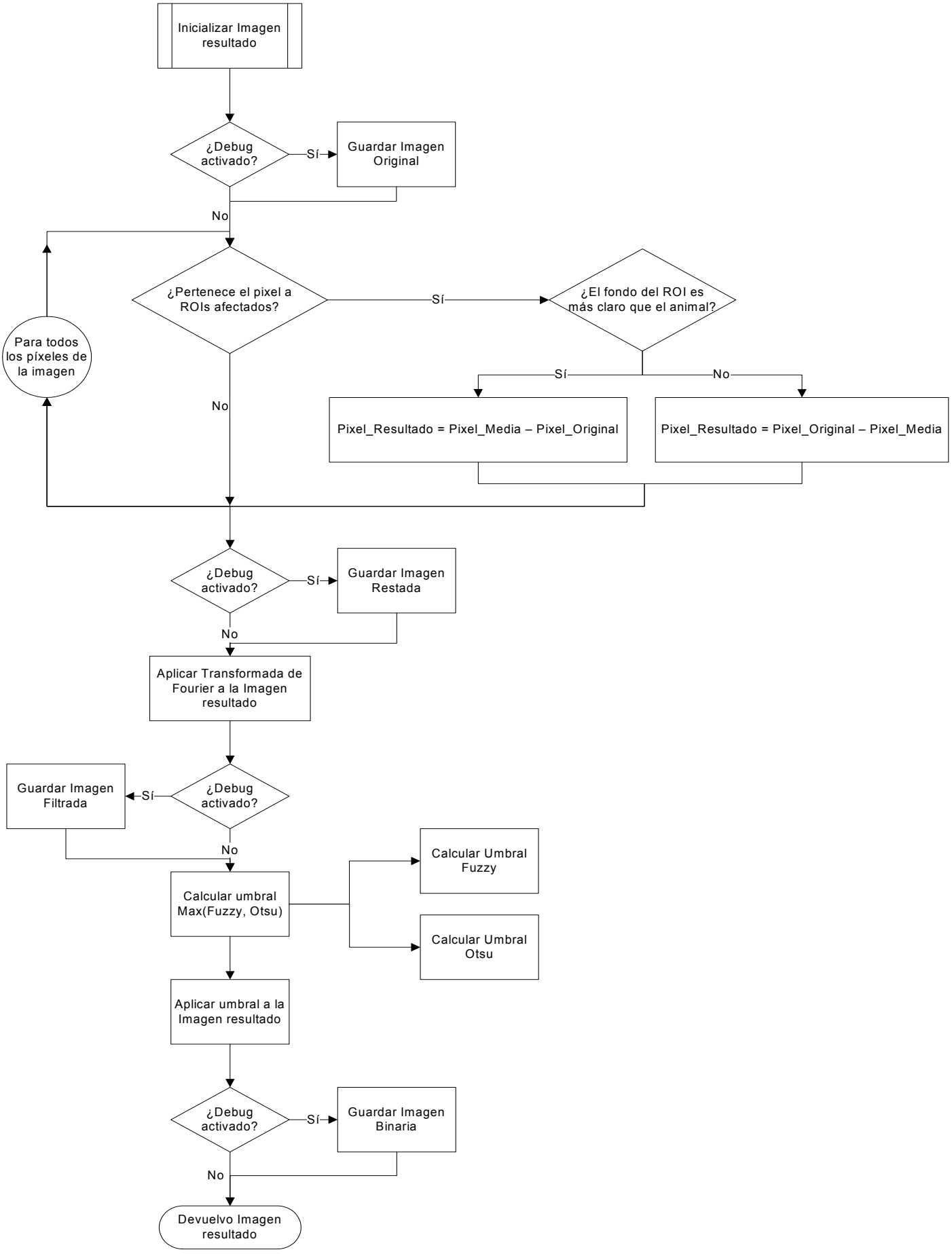


Diagrama de flujo del algoritmo del procesado de la imagen

La primera sentencia que ejecuta el algoritmo es crear una imagen vacía (en negro) que llamaremos la imagen resultado. Esta imagen va a ser la imagen binarizada, la cual devolveremos al algoritmo superior (el que analiza cada imagen y que posteriormente realizará una búsqueda del objeto).

Como se describió anteriormente, el programa se ha preparado para depurar los errores en la detección sin necesidad de modificar el código fuente. Para ello se pueden almacenar las imágenes y analizarlas posteriormente para determinar los posibles errores del algoritmo (comúnmente los errores se producen en los parámetros de tamaño y movimiento). Por ello, el paso siguiente en el algoritmo es guardar la imagen original si la opción *Debug* está marcada.

El siguiente paso que se realiza es una resta de la imagen original con la imagen media en toda la imagen. En un principio se realizaba la resta en toda la imagen, pero más adelante se vio la necesidad de que el algoritmo ganara velocidad en el análisis, por lo cual se decidió procesar sólo la parte interesante de la imagen y no perder tiempo de proceso en localizaciones de la imagen donde el animal no se va a encontrar.

La primera solución que se implementó aplicaba el algoritmo de procesado sólo en los habitáculos, lo cual ganaba mucho tiempo de proceso. Sin embargo, posteriormente se comprobó que lo mejor sería aplicarlo sólo en los ROIs afectados, lo cual fue muy ventajoso para la velocidad del análisis. En apartados posteriores se explicará en profundidad cómo se realiza la búsqueda de los ROIs afectados.

RESTAR IMAGEN ORIGINAL CON LA IMAGEN MEDIA

Con este procedimiento buscamos obtener una imagen en la que el animal destaque en gran medida del fondo. La imagen original es una imagen de los habitáculos con animal, mientras que la imagen media es una imagen tomada del mismo lugar, pero sin animal. Por tanto, teóricamente, restando la imagen original a la imagen media obtendremos una imagen donde sólo esté el animal. El cálculo que se realiza en esta operación es el siguiente:

$$Imagen_{Resultado}(x,y) = Imagen_{Original}(x,y) - Imagen_{Media}(x,y)$$

Si aplicamos esta fórmula a dos píxeles diferentes, uno perteneciente al animal, y otro a la categoría denominada *background* (fondo), podemos observar un aumento de la diferenciación entre las dos categorías:

$$Valor\ Imagen_{Original} = 96 \qquad Valor\ Imagen_{Media} = 100$$
$$Valor\ Imagen_{Resultado} = -4 \rightarrow 0$$

En este caso, hemos aproximado todos los valores negativos a cero. Esto se debe a que el animal es más brillante que el fondo, por lo tanto, si un valor en la imagen original es menor que en la imagen media, se debe a un cambio en el brillo de la sala, no al animal en sí.

Ahora veamos qué pasa si realizamos el mismo proceso en los píxeles en los que tenemos objeto, donde se encuentra el animal:

$$Valor\ Imagen_{Original} = 213 \qquad Valor\ Imagen_{Media} = 99$$
$$Valor\ Imagen_{Resultado} = 114$$

En la siguiente página podemos observar de dónde se han obtenido estos valores, y el resultado que se obtiene en la imagen media.

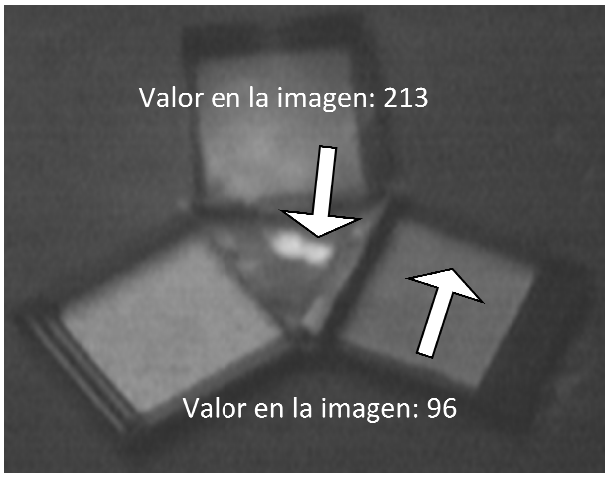


Imagen Original

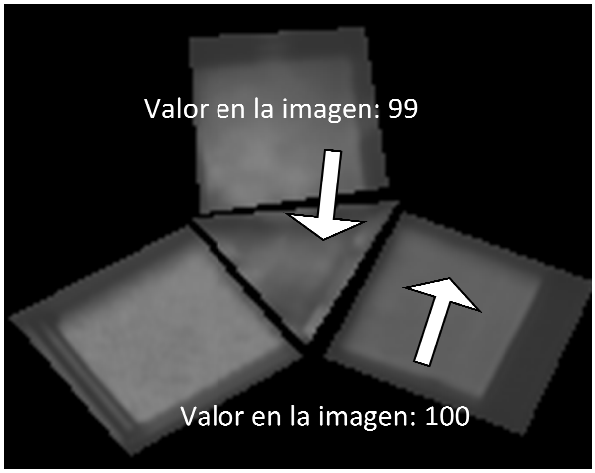


Imagen Media

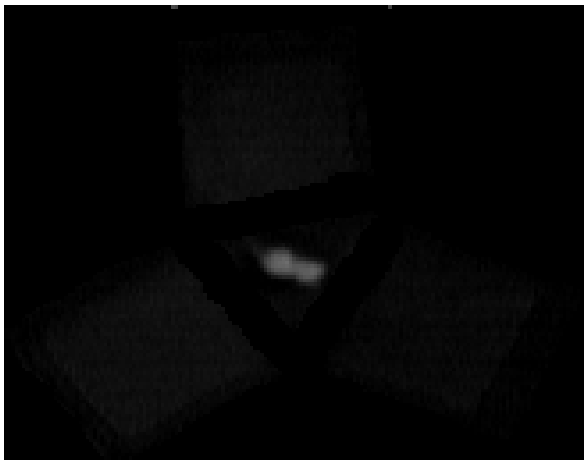


Imagen Resultado

Cuando se dibujaba el habitáculo en la interfaz gráfica era necesario especificar si se trataba de un ROI en el cual el objeto era menos claro que el fondo.

Diálogo para Pintar el Roi

La utilidad de este método se puede considerar en este momento. Si el objeto es menos claro que el fondo en la imagen, al aplicar la resta de la imagen media, eliminaremos el objeto y destacaremos el fondo, es decir, tendremos un *agujero* de color negro en la imagen.

Para solucionar esto, y que el algoritmo de segmentación detecte que es un objeto y no lo confunda con el fondo, lo que se hace es restar a la inversa, de la siguiente manera:

$$Imagen_{Resultado}(x,y) = Imagen_{Media}(x,y) - Imagen_{Original}(x,y)$$

Restando imagen original a media o viceversa, obtenemos una imagen resultado donde la diferencia entre el objeto y el fondo se ha acentuado, lo que favorecerá la detección final del animal.

El siguiente paso es guardar esta imagen restada, para como ya se ha comentado, comprobar, cuando hay errores en la detección, si el problema se debe a esta resta en la imagen media. Por tanto, si *Debug* está activado se pasa a hacer una copia de la imagen resultado en este momento.

El siguiente paso es aplicar un filtro paso banda para seguir con el procesado de la imagen.

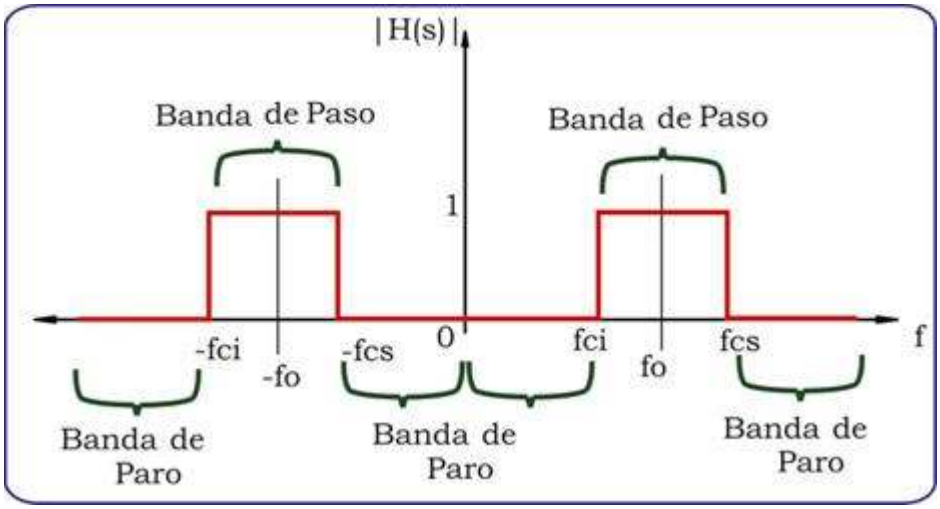
APLICAR FILTRO PASO BANDA

Cuando tenemos la imagen restada, se procede a efectuar un filtrado paso banda en el dominio de la frecuencia a la señal, para reducir el ruido y destacar aún más el objeto buscado.

Un filtro paso banda es un recurso utilizado en el procesamiento de señales por el cual podemos eliminar la parte no deseada de una señal de información, por ejemplo el ruido.

En nuestro caso estamos intentando eliminar los objetos que no tengan un tamaño parecido al que buscamos, para destacar el objeto buscado.

El filtro implementado tiene la siguiente respuesta al impulso:



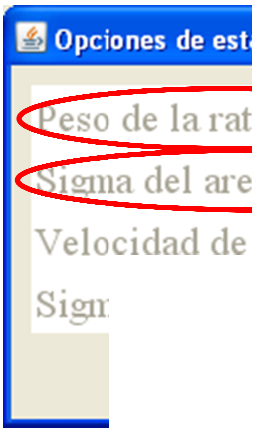
Respuesta al impulso de un filtro paso banda

donde:

El parámetro de va
entre la longitud del anim
ocupa de la imagen, y la pc

Lo

Este valor lo introdu



Se pone de manifiesto en esta imagen que la información que recibimos del usuario no es la longitud del animal, sino el peso. Por ello es necesario convertir esta información (el peso) a una información que podamos utilizar en el filtro.

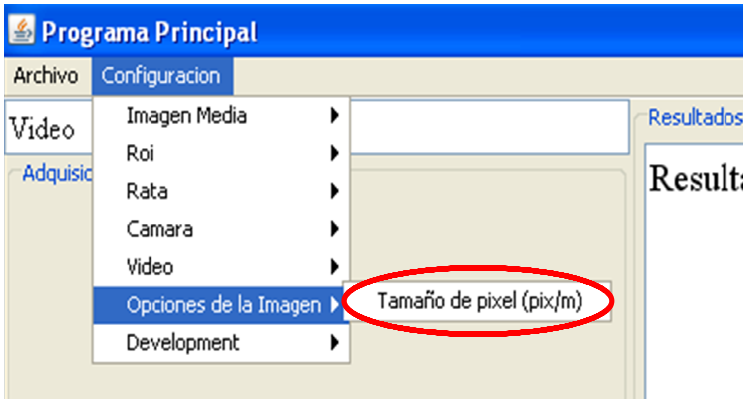
La solución que se ha implementado en el programa es empírica, aprovechando uno de los experimentos llevados a cabo durante el desarrollo del proyecto. Estudiando las imágenes y conociendo el peso de los animales, se ha establecido una tabla con diferentes relaciones entre el peso y la longitud.

PESO (GRAMOS)	LONGITUD DEL ANIMAL (METROS)
250	1,41
275	1,45
300	1,51
325	1,55

Longitud del animal frente al peso

Si ajustamos una ecuación a los datos anteriores obtenemos una relación para calcular la longitud del animal cuando el usuario introduzca el peso. Sin embargo, con esta ecuación estamos calculando el valor de la longitud en metros, mientras que nosotros necesitamos su equivalente en píxeles.

Para ello hacemos uso de otra información que proporcionó el usuario antes de realizar el experimento:



Diálogo para calcular el tamaño real en metros del píxel

Recordemos que siguiendo el proceso que se lanza al marcar esa opción de configuración calculamos la relación píxeles / metro de la imagen. Finalmente, multiplicando este valor por la longitud en metros obtenemos la longitud en píxeles:

$$Longitud_{pixeles} = Longitud_{metros} * Relación \frac{pixeles}{metros}$$

La varianza permanece sin cambios en las unidades, no se le aplica la relación anterior, ya que este valor se introduce en porcentaje.

Tras esta explicación del filtrado y como obtener los parámetros del mismo, vamos a observar los efectos empíricos que se obtienen tras aplicarlos a una imagen

Imagen Restada → Imagen Filtrada

En la imagen restada el fondo comprende el rango de valores de 0 a 10, mientras que el animal tiene un valor aproximado de 100. En la nueva imagen, la filtrada, el fondo está entre 0 y 50. Sin embargo el valor del animal en este caso es de 220.

Hemos conseguido por tanto destacar el objeto sobre el fondo más de lo que ya estaba, que era el fin que buscábamos.

El siguiente paso en nuestro algoritmo es de nuevo almacenar esta imagen si así lo ha requerido el usuario marcando la opción *Debug*, y tras ello la umbralización.

CÁLCULO DEL UMBRAL

La parte más importante de la umbralización para poder aplicar el siguiente método es calcular de forma acertada el umbral:

$$Pixel\ nueva\ imagen(x,y) = \begin{cases} 1 \Leftrightarrow Pixel\ imagen(x,y) \geq Umbral \\ 0 \Leftrightarrow Pixel\ imagen(x,y) < Umbral \end{cases}$$

El umbral se calcula mediante el análisis del histograma de la imagen. En este momento es necesario explicar en qué consiste un histograma:

El histograma de una imagen digital con niveles de gris en el rango [0, 255] es una función discreta $p(r_k)= n_k/n$, donde r_k es el k-ésimo nivel de gris, n_k es el numero de píxeles de la imagen con ese nivel de gris y n el número total de píxeles de la imagen (k = 0, .., 255). En forma general podemos decir que $p(r_k)$ da una idea acerca de la probabilidad de que aparezca el nivel de gris r_k. La representación grafica de esta función para todos los valores de k proporciona una descripción global de la apariencia de la imagen. Aunque el histograma no indica nada específico sobre el contenido de la imagen, su perfil proporciona sin duda información muy útil sobre la posibilidad de mejorar la imagen, entendiendo por mejora el destacar características de interés de la misma y no en el sentido usual visual.

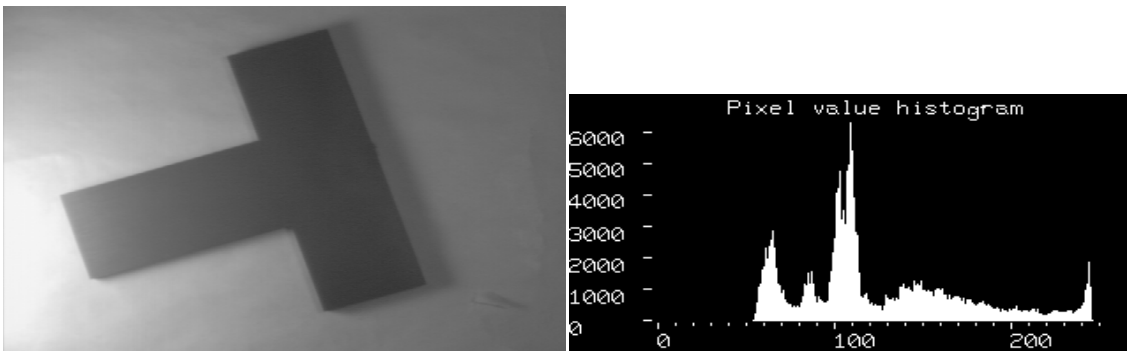


Imagen ejemplo y el histograma asociado

Podemos observar en la imagen que hay pocos valores de negro. Eso se puede ver claramente en el histograma, donde el número de píxeles con valor 50 ò menos es muy inferior al resto. Nótese además el pico de píxeles con intensidad cercana a 255 (blanco) que se puede localizar a la izquierda de la imagen.

En el siguiente ejemplo podemos observar el histograma de la imagen original y el histograma de la imagen después del procesado:

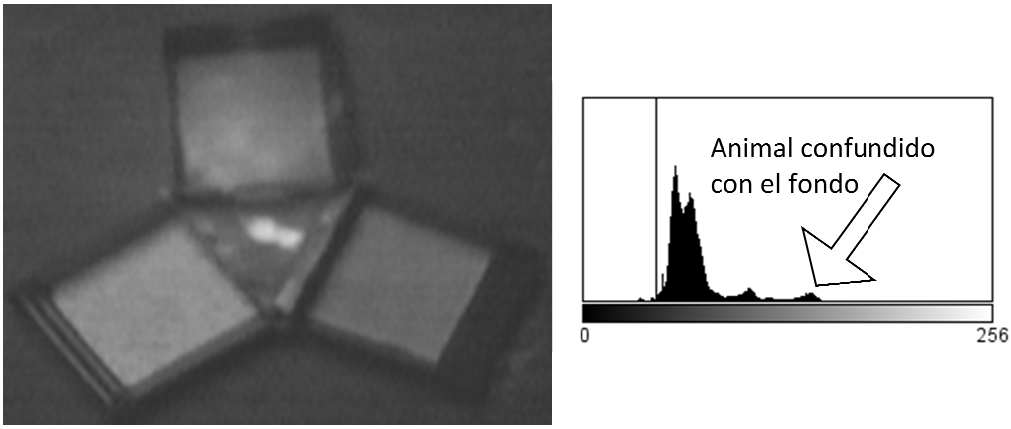


Imagen Original y su histograma

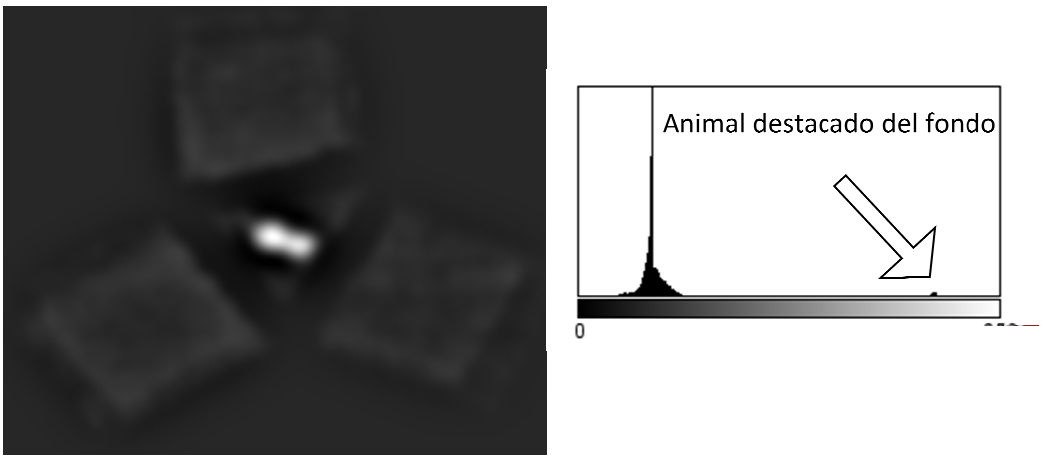


Imagen procesada (resta y filtro) y su histograma

En la segunda imagen es más sencillo determinar el umbral, y el error cometido va a ser menor (gracias al procesado previo de resta y filtro). Si en la segunda imagen

Por tanto, suponiendo que el animal es la parte más clara de la imagen, podemos establecer los límites de la siguiente forma:

Límite superior = 1
Límite inferior = 1 – número de píxeles del animal

Con los límites establecidos, el algoritmo se puede ejecutar sin más intervención del usuario. Huelga decir que esta operación es totalmente transparente para el usuario, ya que el número de píxeles del animal se define antes de la realización del análisis, y es un parámetro que se calcula automáticamente con el peso del animal, de manera análoga al cálculo de la longitud del animal.

Lo siguiente que realiza el algoritmo es dividir el tramo establecido entre los límites en cuatro partes. De cada una de estas divisiones se calcula la entropía del objeto y del fondo, y nos quedamos con el tramo con mayor entropía.

Finalmente, el umbral es el punto medio de este tramo.

En la siguiente página podemos ver el gráfico que describe este método, tal y como se ha implementado en el proyecto.

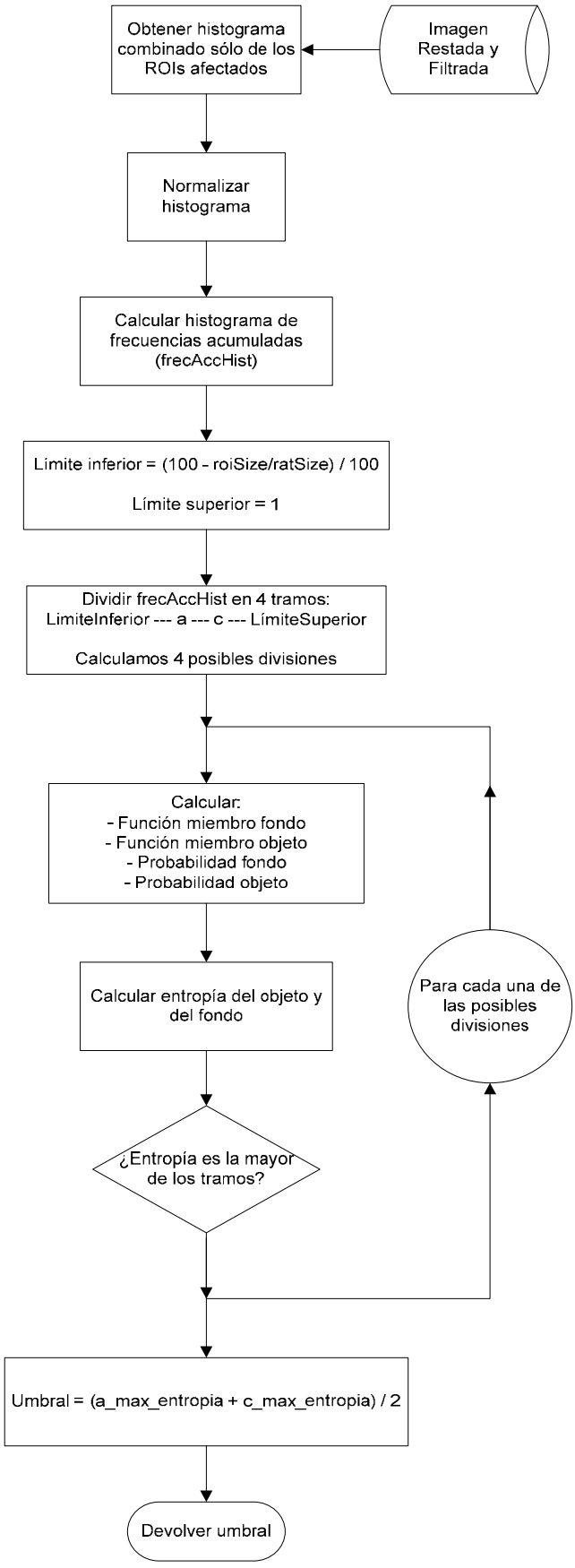


Diagrama de flujo del algoritmo de umbralización de Fuzzy

Después de calcular los dos umbrales, nos quedamos con el mayor de ellos, ya que como se ha dicho anteriormente, es conocido que el animal es la parte más blanca de la imagen, y empíricamente se ha demostrado que el umbral tiene problemas de incluir más fondo que objeto, por lo que se puede aplicar el máximo umbral entre ellos sin ningún problema.

Siguiendo con el algoritmo principal de binarización, ya hemos conseguido una imagen con dos únicos niveles: negro para el fondo y blanco para el objeto. Lo siguiente que hacemos en nuestro algoritmo es almacenar una copia de esta imagen si así nos lo indica *Debug*. Tras esto devuelvo la imagen binarizada al método principal de detección, que se encargará de buscar los posibles objetos que se encuentren en la imagen.

Resumiendo, el algoritmo de binarización realiza dos procesos bastante diferenciados. En el primero, mediante técnicas de restado filtrado de la imagen (procesado digital), se intenta aumentar la diferencia en el histograma entre el objeto buscado y el fondo en el que se encuentra.

La segunda parte del algoritmo es aplicar un umbralizado para binarizar la imagen. Para éste umbralizado se calculan dos umbrales (*threshold*) diferentes, para posteriormente aplicar el mayor de ellos.

DETECTAR ANIMAL EN LA IMAGEN BINARIZADA

En estas últimas páginas hemos estado desarrollando el proceso por el cual obtenemos una imagen binarizada, donde el animal tiene un valor en la imagen, y el fondo otro distinto, pero sólo hay dos valores.

Si este proceso fuese totalmente determinista no encontraríamos ningún error. La imagen siempre contendría un único objeto y el fondo. Sin embargo esto no es así. El animal se mueve, se esconde, se sitúa entre un habitáculo y otro, la cámara percibe ruido, etc.

Son muchas las posibles situaciones que pueden hacer que el procesado digital de la imagen no binarize de manera determinista, y nos encontremos casos en los que la imagen detecte más de un objeto, o casos en los que se procese toda la imagen como si fuera fondo.

Por todo esto se hace necesario un algoritmo robusto que procese la información que nos devuelve el análisis de la imagen y que determine entre varios objetos cual es el animal y cual es el ruido, y en el caso de que no detecte ningún objeto tratar ese error de manera eficiente.

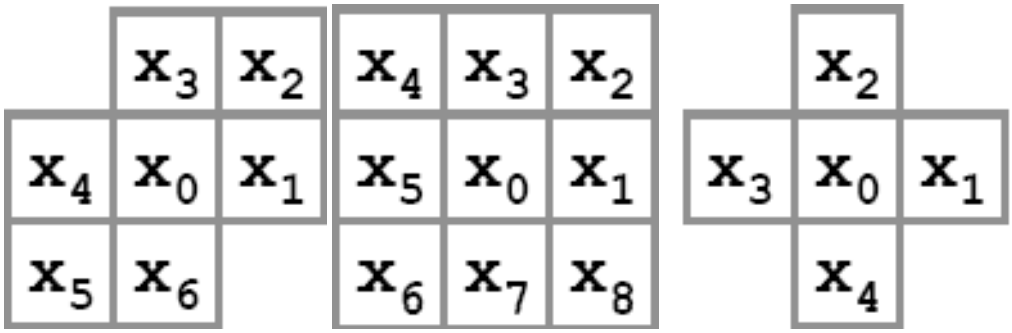
Pero antes de explicar como se ha implementado esta detección, recordemos en qué parte del algoritmo principal nos encontramos. Habíamos tomado la imagen de la fuente, la hemos analizado y tenemos la imagen binaria.

Antes de decidir entre los objetos se hace necesario definir cuales son estos objetos, es decir, crear un nuevo vector o array donde almacenar las coordenadas de cada punto de la imagen que pertenezcan a ese objeto.

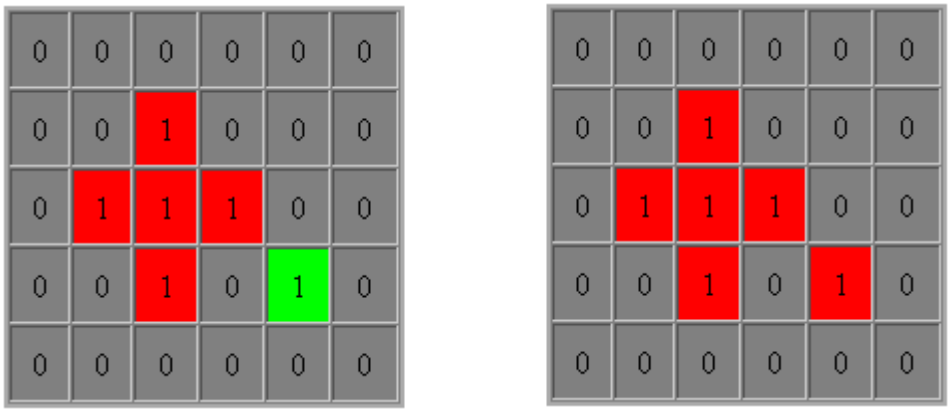
Para ello hay que definir y eligir entre varios métodos de conectividad, pero antes explicaremos cual es el concepto de conectividad

CONECTIVIDAD

Cuando se habla de píxeles conectados entre sí o yuxtapuestos, es necesario precisar el tipo de conexión o conectividad que se acepta como válida. En principio, un píxel de una malla rectangular puede estar conectado con los ocho píxeles que le rodean en un espacio plano o solamente con los cuatro más cercanos. Ello dará lugar a hablar de una conexión de tipo 8, de tipo 4, o incluso de tipo 6.



Conectividad – 4 Conectividad – 8 Conectividad - 6



Ejemplo conectividad - 4 y conectividad - 8

Los píxeles marcados en rojo forman un objeto con una conexión de tipo 4, mientras que el píxel marcado en verde formaría un nuevo objeto con este tipo de conexión. Por el contrario, si la conexión admitida es de tipo 8 (incluye los píxeles que se conectarían teóricamente sólo por una esquina), los cinco píxeles con valor verdadero (1) en el ejemplo formarían un solo objeto.

Para este proyecto se ha decidido implementar el método de conectividad – 8, ya que empiricamente se ha demostrado que es más efectivo. Podemos ver a continuación el diagrama de flujo del algoritmo desarrollado:

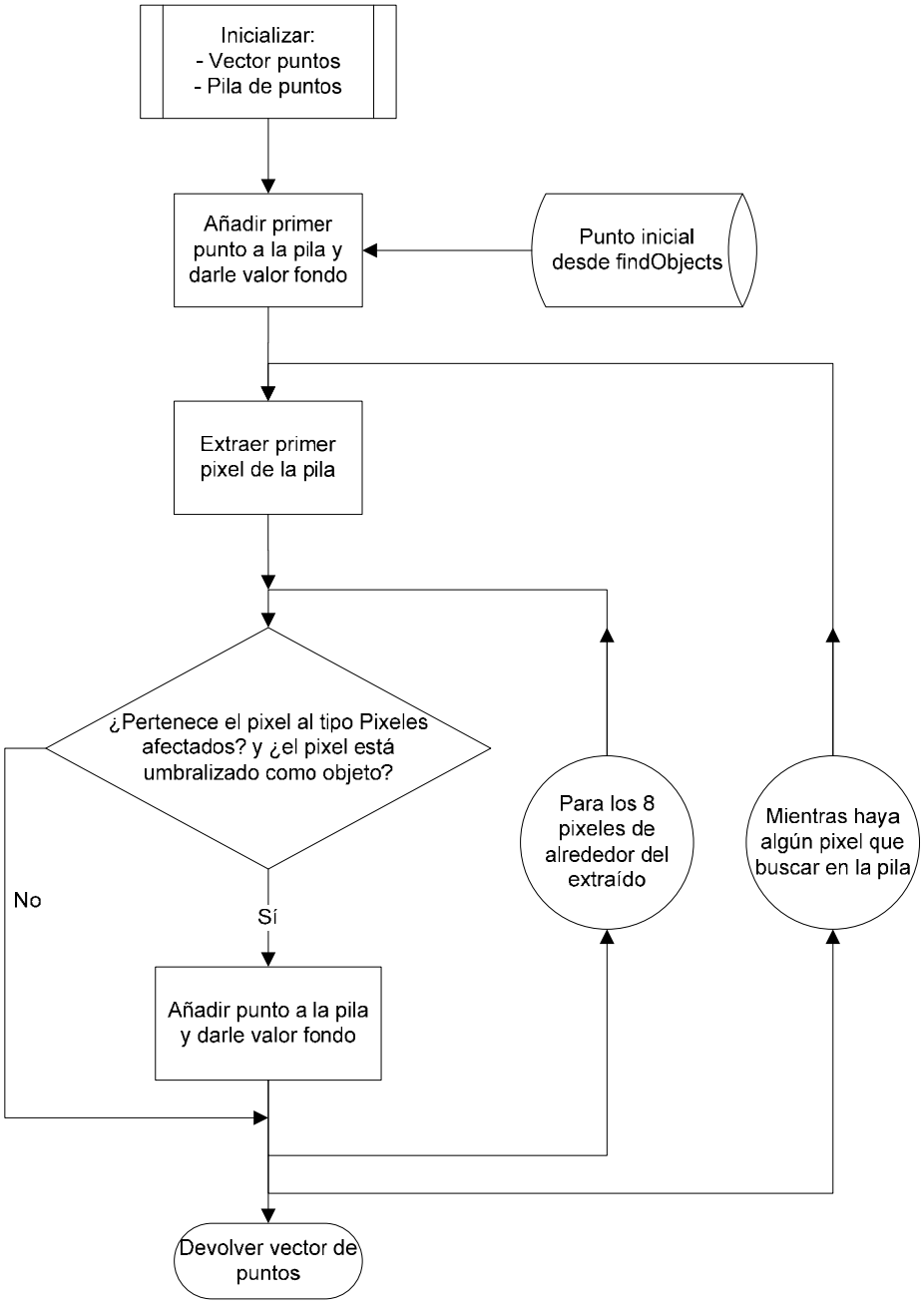


Diagrama de flujo conectividad - 8

Este algoritmo es llamado desde la función *findObjects*, que es la encargada de buscar objetos en la imagen procesada. Esta función le pasa como parámetro al método de conectividad un único punto perteneciente al objeto a conectar. Es decir, se busca en la imagen binarizada el valor del tipo objeto, y le pasamos a este método ese punto y la imagen a buscar.

El primer paso que realiza este algoritmo es inicializar el vector puntos, que es el vector donde se van a almacenar todos los puntos pertenecientes a este objeto. También inicializa una pila (vector de tipo primero que entra último en salir (LIFO)) que se va a usar en el desarrollo del algoritmo.

Añadimos el primer punto a la pila y al vector de puntos, y le damos el valor de tipo fondo a ese pixel en la imagen. Cada vez que añadamos un punto a la pila y al vector, cambiaremos su valor en la imagen de tipo objeto a tipo fondo. De esta manera estaremos asegurando que ese pixel no se vuelve a introducir en la pila ni en el vector de puntos.

Extraemos el primer elemento de la pila, y comprobamos si los 8 pixeles adyacentes (los yuxtapuestos) son de tipo fondo o de tipo objeto. Si son de este último tipo los añadiremos a la pila y al vector de puntos, y le daremos valor del tipo fondo. Y volveremos a repetir la operación mientras haya puntos en la pila.

Hay que añadir que se comprueba otra condición antes de añadir a la pila. La comprobación que se realiza es si el pixel pertenece al habitáculo, para no añadir nada que no se encuentre dentro.

En un principio esta comprobación se realizaba mediante una función que comprobaba si el punto pertenecía a uno de los polígonos que conformaban los ROIs. Sin embargo, la ejecución de esta función no era inmediata, y realizaba bastantes operaciones, ya que tenía que extraer cada polígono del vector donde se contenían los ROIs, y para cada uno de ellos preguntar (mediante una función costosa de proceso) si el pixel (sus coordenadas) estaba contenido en el área del polígono.

Para evitar toda esta carga de proceso que enlentencía en gran medida el proceso de detección, se decidió crear una imagen paralela a la original. Cuando el usuario dibuja los ROIs o los carga desde un archivo, una función se encarga de crear una imagen en negro, y sobre ella dar un valor preestablecido a todos los píxeles que pertenecen a un ROI.

De esta manera, cuando se quiere comprobar si un pixel de la imagen está dentro de un ROI o fuera, sólo hay que comprobar si el valor de este pixel en la imagen paralela tiene valor de ROI o no. Este proceso es casi inmediato, y se consiguió reducir muchísimo el tiempo de proceso del algoritmo.

Finalmente el método devuelve el vector de puntos del animal.

Resumiendo, el algoritmo descrito recibe un punto y una imagen, y busca siguiendo el método de conectividad – 8 todos los puntos que pertenecen al objeto, y cuando termina nos devuelve el vector de puntos, es decir, un vector que contiene todos los puntos que se consideran del objeto.

PROCESAR LA POSICIÓN

Ya hemos descrito la técnica con la cual conectar los pixeles de una imagen para conformar un objeto, pero aún no hemos explicado el algoritmo por el cual de una imagen binarizada obtenemos la posición del animal, qué pasa cuando hay más de un objeto en la imagen, etc. Para ello hacemos uso del siguiente gráfico de flujo, donde se explica el procedimiento:

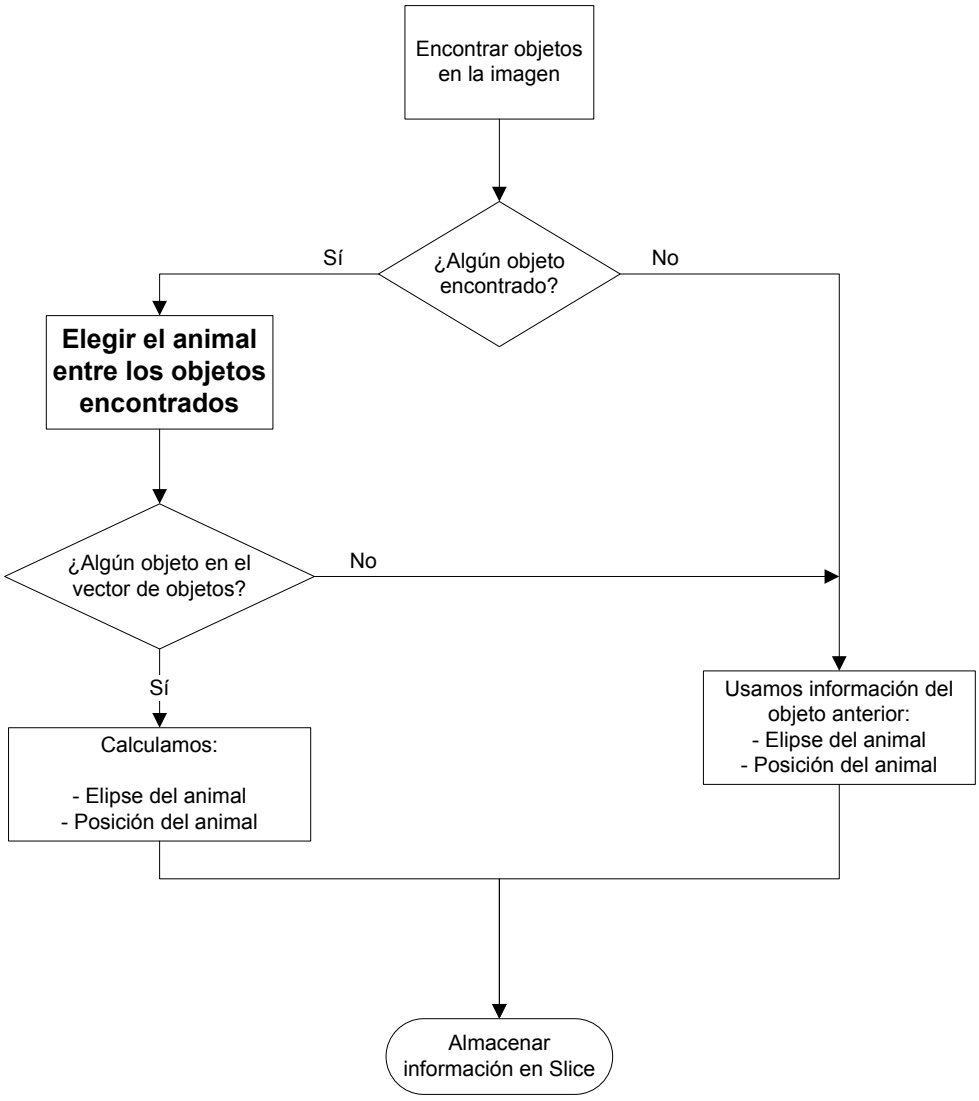


Diagrama de flujo del algoritmo procesar posición

Como podemos observar, el primer paso que realiza el algoritmo es encontrar objetos en la imagen. El procedimiento de buscar objetos en la imagen es el método *findObjects*, que es el encargado de buscar el primer punto de un objeto y llamar a la técnica de conectividad. Este método también necesita de su propio diagrama de flujo para explicar el funcionamiento, el cual podemos observar en la siguiente página.

Este método tiene como primera sentencia la inicialización del vector de objetos, además de comprobar que ya se ha inicializado la imagen de pixeles afectados, y los valores xMin, xMax, yMin e yMax. El uso de la imagen de pixeles afectados ya se ha explicado, por lo que sólo se explicará que significan los valores xMin, xMax, yMin e yMax.

La búsqueda de objetos en la imagen se tiene que llevar a cabo en todos los píxeles, que dependiendo del tipo de imagen captada el número puede ser mayor o menor, en nuestro caso 640x480=307200 píxeles donde buscar.

Sin embargo, es muy normal encontrarnos imágenes donde los ROIs conforman sólo un 60% del total de la imagen, a veces incluso menos. Para hacer nuestro algoritmo aún más efectivo en términos de velocidad se decidió ajustar un cuadrado lo más pequeño posible alrededor de los habitáculos en la imagen, y buscar objetos dentro de este cuadrado.

Por ello, y de la misma forma que se crea la imagen de píxeles afectados por el ROI, cuando el usuario dibuja el ROI o cuando se carga de archivo, se llama a un método que ajusta el cuadrado más pequeño a este dibujo y nos devuelve las cuatro esquinas del cuadrado (xMin, xMax, yMin e yMax).

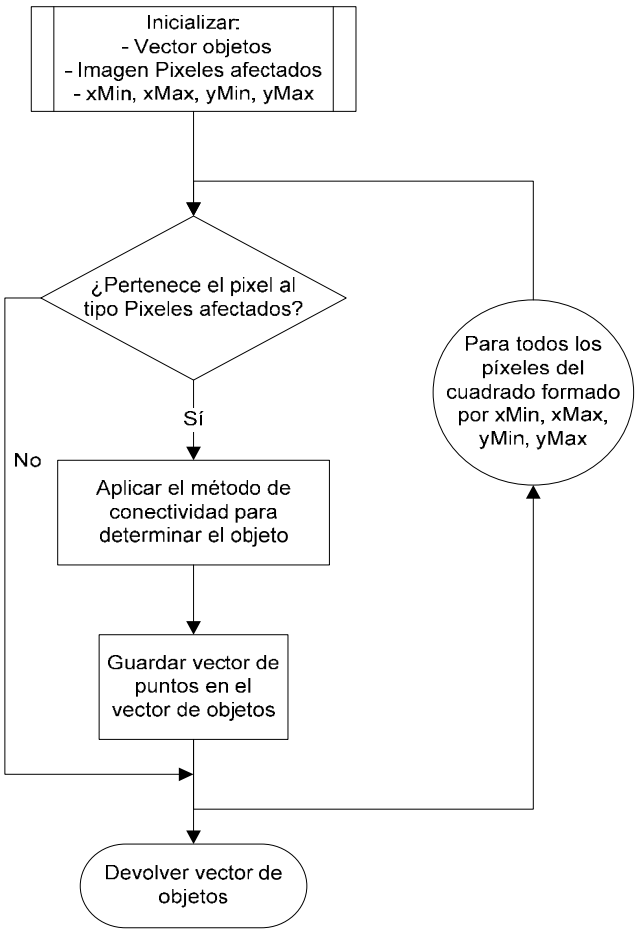


Diagrama de flujo del algoritmo para encontrar objetos en la imagen

Después de inicializar el vector y estas variables, se busca dentro de este cuadrado puntos del tipo objeto y que pertenezcan a los ROIs (tipo pixel afectado). Cuando se encuentra uno de estos puntos se llama al algoritmo de conectividad, que nos devolverá el vector de puntos que conforma el objeto y lo almacenaremos en nuestro vector de objetos.

Resaltar que el método de conectividad ha cambiado el valor de los pixeles de la imagen binarizada que ha determinado que son del mismo objeto, de tipo objeto se cambiarán a tipo fondo (de 255 a 0), para que, de esta manera, cuando se vuelva a buscar en la imagen no se añada de nuevo el mismo objeto.

Es decir, este algoritmo llamará al método de conectividad una vez para cada objeto en la imagen, ya que el método de conectividad se encarga de "ocultar" el objeto en la imagen cuando lo añade a su vector de puntos.

Recapitulando, tenemos un algoritmo que es llamado tras binarizar la imagen y que se encarga de buscar objetos en ésta, y tras unirlos siguiendo el método de conectividad nos devuelve un vector con todos ellos, entre los cuales está el animal.

ELECCIÓN DE OBJETOS

El siguiente paso en nuestro algoritmo de procesamiento de la posición del animal es comprobar si el vector de objetos tiene algún objeto o si por el contrario se encuentra vacío.

No encontrar ningún objeto en la imagen casi nunca sucede, pero puede darse el caso, por lo cual se contempla en el algoritmo. En este supuesto, el algoritmo usará la información del instante anterior, por tanto almacenaremos en el *slice* la información del *slice* anterior, y marcaremos este *slice* como problemático, para, como ya se dijo anteriormente, proceder a un re análisis posterior del instante de tiempo.

Cuando el vector objetos tiene un objeto o más almacenados procedemos a elegir cual de ellos es el animal mediante el algoritmo de elección. Este algoritmo modificará el contenido del vector de objetos, y nos devolverá un objeto o ninguno.

El objeto devuelto es el animal, por lo cual calcularemos la elipse del animal y su posición, y almacenaremos esta información en el objeto *slice*.

En el caso de no devolver ningún objeto, de nuevo se usará la información del instante de tiempo anterior y se volverá a marcar este *slice* como problemático.

El algoritmo de elección de objetos se encuentra desarrollado a continuación en un diagrama de flujo que facilita su comprensión.

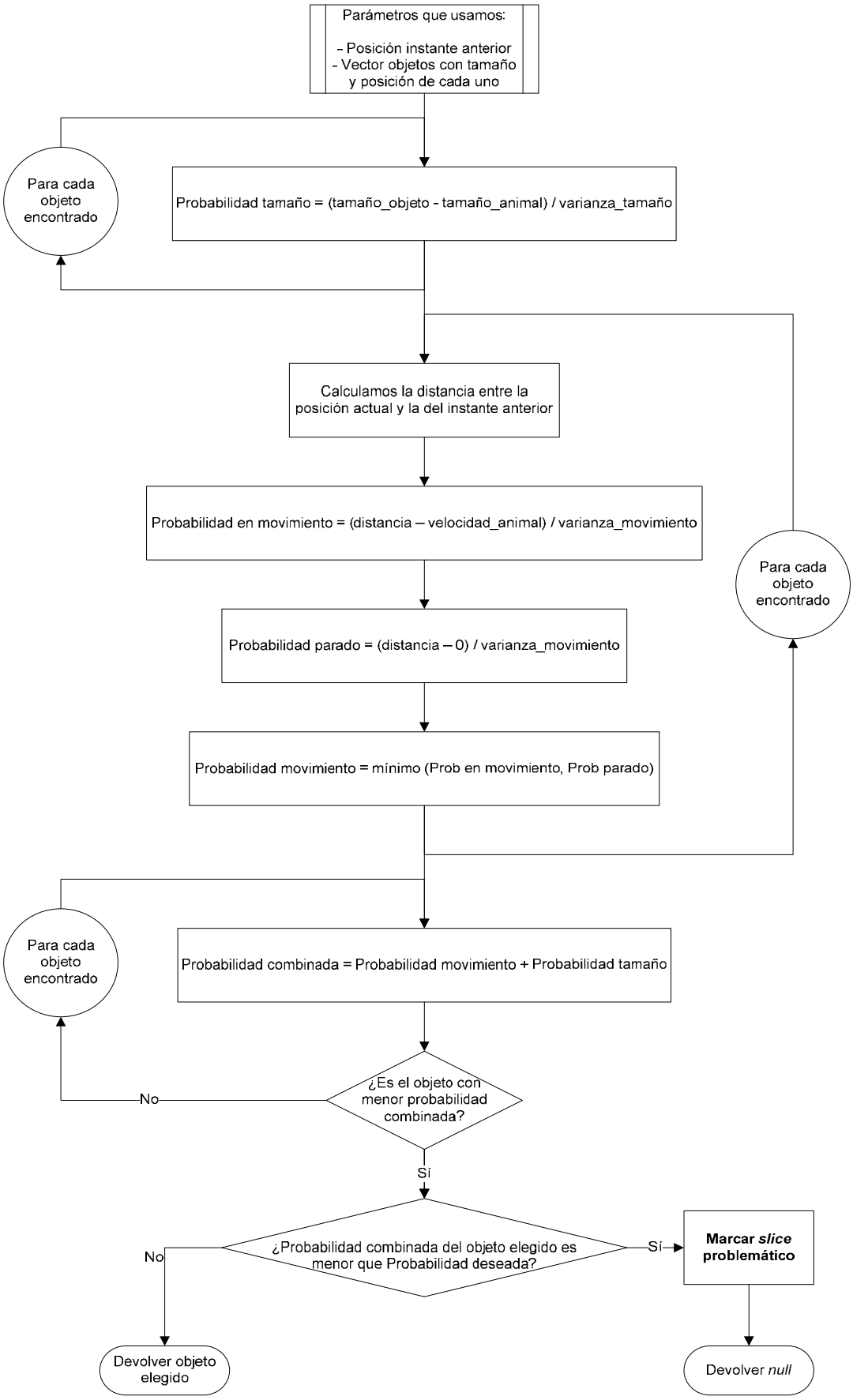


Diagrama de flujo del algoritmo de elección de objetos

Para ilustrar por qué se usa un algoritmo de elección de objetos se recomienda observar la siguiente imagen, producto de una mala umbralización:

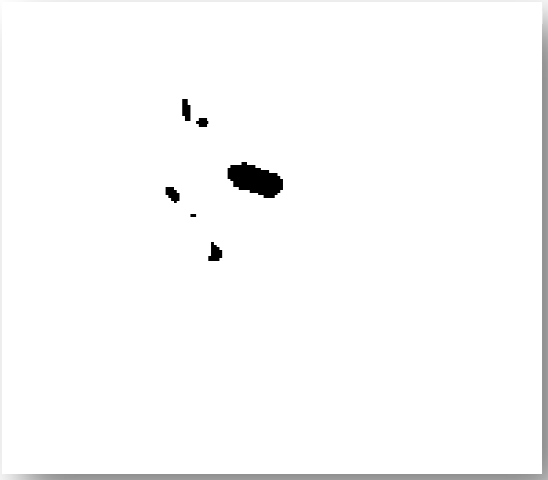


Imagen mal umbralizada

En esta imagen podemos observar el animal buscado (el objeto central), además de otros objetos que no pertenecen al objeto buscado, sino que son objetos ruidosos producto de la mala umbralización.

Tras unir cada objeto mediante los algoritmos descritos anteriormente, es el momento de que el programa tome de manera automática una decisión, sin que el usuario tenga ningún tipo de intervención. Para ello vamos a hacer uso de los conocimientos previos que conocemos sobre el animal.

Los parámetros que conocemos sobre este animal son el peso y la velocidad máxima. Gracias al peso podemos calcular cual debe ser el tamaño en pixeles del objeto; y conociendo la velocidad máxima podemos determinar la distancia máxima a la que se puede encontrar el animal entre una instante de tiempo y el siguiente.

Por tanto, conociendo el valor que debe tomar una variable, y una cierta varianza, podemos calcular la distribución normal de la función peso y de la función velocidad para obtener los valores más usuales y cuales son menos probables. De esta manera podemos cuantificar la probabilidad de que un objeto sea el animal buscado, y tomar gracias a ello una decisión.

Esto es lo que hacemos en el algoritmo anterior. En primer lugar recogemos la posición del instante anterior, además de tener el vector de objetos con el vector de puntos de cada uno de ellos, de donde podemos calcular la posición y el tamaño del objeto (número de píxeles que conforman el objeto).

El siguiente paso consiste en calcular la probabilidad por tamaño de cada objeto que se encuentre en el vector siguiendo la fórmula de la distribución normal:

$$Z = \frac{X - \mu}{\sigma} \quad \rightarrow \quad Probabilidad_{tamaño} = \frac{tamaño_{objeto} - tamaño_{animal}}{varianza\ tamaño}$$

Posteriormente tenemos que calcular la probabilidad por movimiento del animal. Para esto necesitamos la posición en el instante anterior, y con la actual calcular la **distancia** entre un instante y otro.

En el caso de probabilidad por movimiento pueden ocurrir dos situaciones: Que el animal se encuentre en movimiento o que el animal esté parado. Por ello calcularemos dos probabilidades y elegiremos la menor Z, es decir el valor más probable:

$$Probabilidad_{movimiento} = \frac{distancia - velocidad}{varianza\ velocidad}$$

$$Probabilidad_{parado} = \frac{distancia - 0}{varianza\ velocidad}$$

Finalmente calcularemos la probabilidad combinada de cada objeto, que será resultado de la suma de la probalidad por tamaño y la probabilidad por movimiento o parado. El objeto animal será aquel que tenga la menor probabilidad combinada (la sumada).

Sin embargo hay ocasiones en las que ninguno de los objetos encontrados son el animal, ya que éste se encuentra escondido entre las puertas o ha ocurrido un problema. Para constatar que ha habido un error se compara la probabilidad combinada del objeto elegido con una probabilidad determinada (un margen de error). Si la probabilidad combinada es mayor que este margen se marca el *slice* como problemático y se devuelve *null*, es decir, un vector vacío.

En el caso de que la probabilidad combinada sea menor que este margen de error o probabilidad deseada se devuelve el vector objetos con un único vector de puntos: el animal.

Si resumimos todo lo que hemos explicado en este apartado podemos decir lo siguiente: La binarización de la imagen procesaba ésta de tal manera que nos entregaba una imagen con dos únicos valores: el tipo objeto y el fondo. Explicamos en primer lugar un algoritmo de conectividad para unir los pixeles y formar objetos.

Tras esto describimos como se busca la posición del animal en la imagen entre estos objetos. En primer lugar buscamos los objetos en la imagen. Si no se encuentra ninguno utilizamos la información del instante anterior y marcamos el *slice* como problemático.

Con los objetos encontrados tenemos que decidir cual de ellos es el animal, para lo cual usaremos la información proporcionada por el usuario sobre el animal y la técnica estadística de distribución normal de la probabilidad para elegir cual de los objetos tiene más probabilidades de ser el animal.

Es probable que el objeto encontrado sea el animal, pero hay casos en los que no es así, y para decidir esto se comprueba si la probabilidad del objeto es mayor que una deseada, caso en el cual no se devuelve ningún objeto y se marca el *slice* como problemático.

DETERMINAR POSICIÓN EXACTA DEL ANIMAL

Tras obtener el vector de puntos que conforman el animal en la imagen procedemos a ajustar una elipse sobre estos puntos. Se ajusta una elipse ya que el animal usado en los experimentos es una rata, que en una vista de planta (desde arriba) tiene una forma elipsoidal.

Finalmente debemos decidir qué punto exacto es el animal. En una primera aproximación al resultado final este punto se consideraba que era el centro de la elipsoide. Sin embargo un hecho que no conocíamos a priori nos hizo cambiar esta solución.

En uno de los experimentos llevados a cabo y mientras se probaba el programa con él, el animal se posicionó entre las puertas de conexión de dos habitáculos, de tal manera que el 80% del cuerpo del animal se encontraba en un habitáculo, mientras que

el resto (incluida la cabeza) se encontraba en la puerta, pero de tal manera que el animal se encontraba mirando el otro habitáculo.

Como hemos explicado en el apartado **Metodología Experimental para el estudio de las drogodependencias**, durante el proceso de CPP no importa dónde se encuentre el animal, sino qué habitáculo llama su atención. Y en ese caso específico le llama la atención el habitáculo hacia donde está mirando.

Por ello se implementó un algoritmo para determinar donde se encuentra la cabeza del animal (en uno de los extremos de la elipse). El diagrama de flujo del algoritmo lo podemos encontrar en la siguiente página.

Como se ha descrito antes, primero ajustamos una elipse con el vector de puntos que nos devuelve la elección del objeto. Para ello se necesitan tres parámetros de la elipse: el punto central, el ángulo theta y la longitud del eje mayor. Con estos parámetros, y basándonos en la ecuación paramétrica de la elipse se pueden calcular los dos extremos de la elipse, que son los puntos que estamos buscando:

$$x_{positivo} = x_{centro} + \frac{eje_{mayor}}{2} * \cos(2 * \pi - \theta)$$

$$y_{positivo} = y_{centro} + \frac{eje_{mayor}}{2} * sin\ (2 * \pi - \theta)$$

$$x_{negativo} = x_{centro} - \frac{eje_{mayor}}{2} * \cos(2 * \pi - \theta)$$

$$y_{negativo} = y_{centro} - \frac{eje_{mayor}}{2} * sin\ (2 * \pi - \theta)$$

Tras conocer los dos puntos de los extremos de la elipse es necesario decidir cuál de ellos corresponde a la cabeza del animal. Para ello hacemos uso de la posición del animal en el instante anterior. Si el animal se mueve en el sentido de su cabeza (hacia delante), entre el instante anterior y el presente, el punto más alejado de la posición anterior será la cabeza, mientras que el más cercano corresponderá a los cuartos del animal.

Este proceso se puede observar cómo se lleva a cabo en la siguiente imagen que describe su diagrama de flujo.

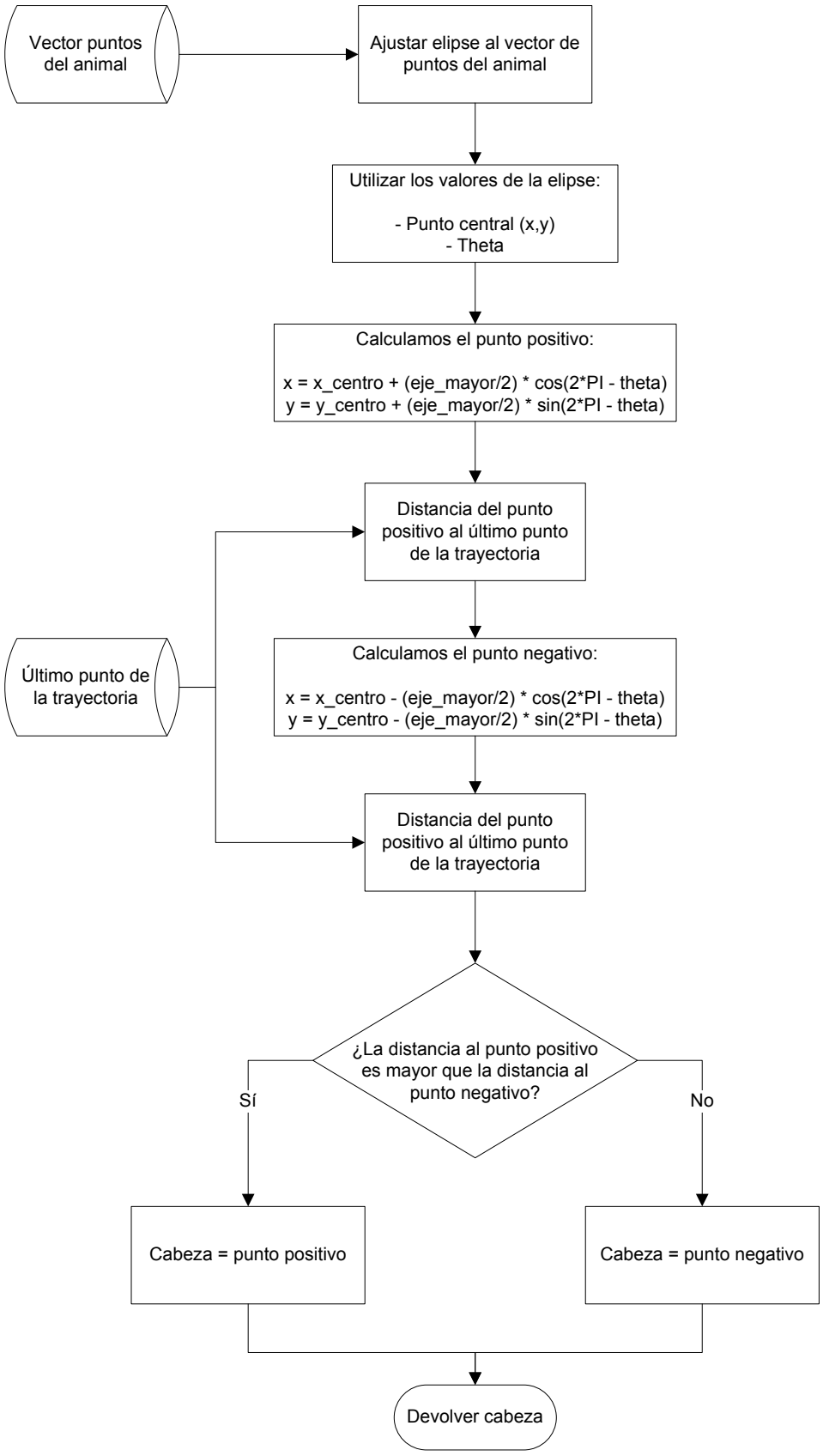


Diagrama de flujo del algoritmo para detectar la cabeza del animal

PROTECCIÓN FRENTE A ERRORES

Hasta ahora hemos explicado cómo detectar la posición del animal y hemos hablado de detectar cuando se ha producido un error. Sin embargo aún no se han definido las acciones a efectuar cuando se produce este error.

El principal problema cuando se produce un error en la detección del animal en un instante determinado es que no se conoce la actuación del animal en este instante; si se ha movido o se ha quedado parado.

Esto modifica los parámetros estadísticos de movimiento para la búsqueda del animal en el siguiente instante. Es decir, es necesario de cara al siguiente instante modificar el radio del círculo donde se puede detectar al animal, y hacerlo más grande.

Pero esto no es suficiente, ya que también se hace necesario aumentar en un 5% el parámetro de varianza del movimiento. Recordemos que esta varianza es una medida de la variación posible en la distribución normal. Esto se puede explicar de la siguiente manera: el animal se mueve normalmente a 10 cm/s, pero es posible que se mueva a más o menos velocidad. La varianza (*sigma*) tiene en cuenta esta posible variabilidad para calcular la probabilidad por movimiento.

Y todo esto se ha tenido en cuenta en las últimas sentencias del algoritmo de detección de la posición del animal, tal y como se observa en el siguiente gráfico de flujo.

En él podemos observar que tras determinar el habitáculo en el que se encuentra el animal se ha colocado un bloque de decisión. En este bloque se comprueba si ha habido problemas en la detección (si no se ha detectado ningún objeto, y si se ha detectado y la probabilidad combinada es mayor que la establecida).

Si ha habido problemas en la detección se llevan a cabo las operaciones descritas de aumento del radio de búsqueda y de aumento de la varianza de la probabilidad de movimiento.

Observémoslo en el siguiente gráfico:

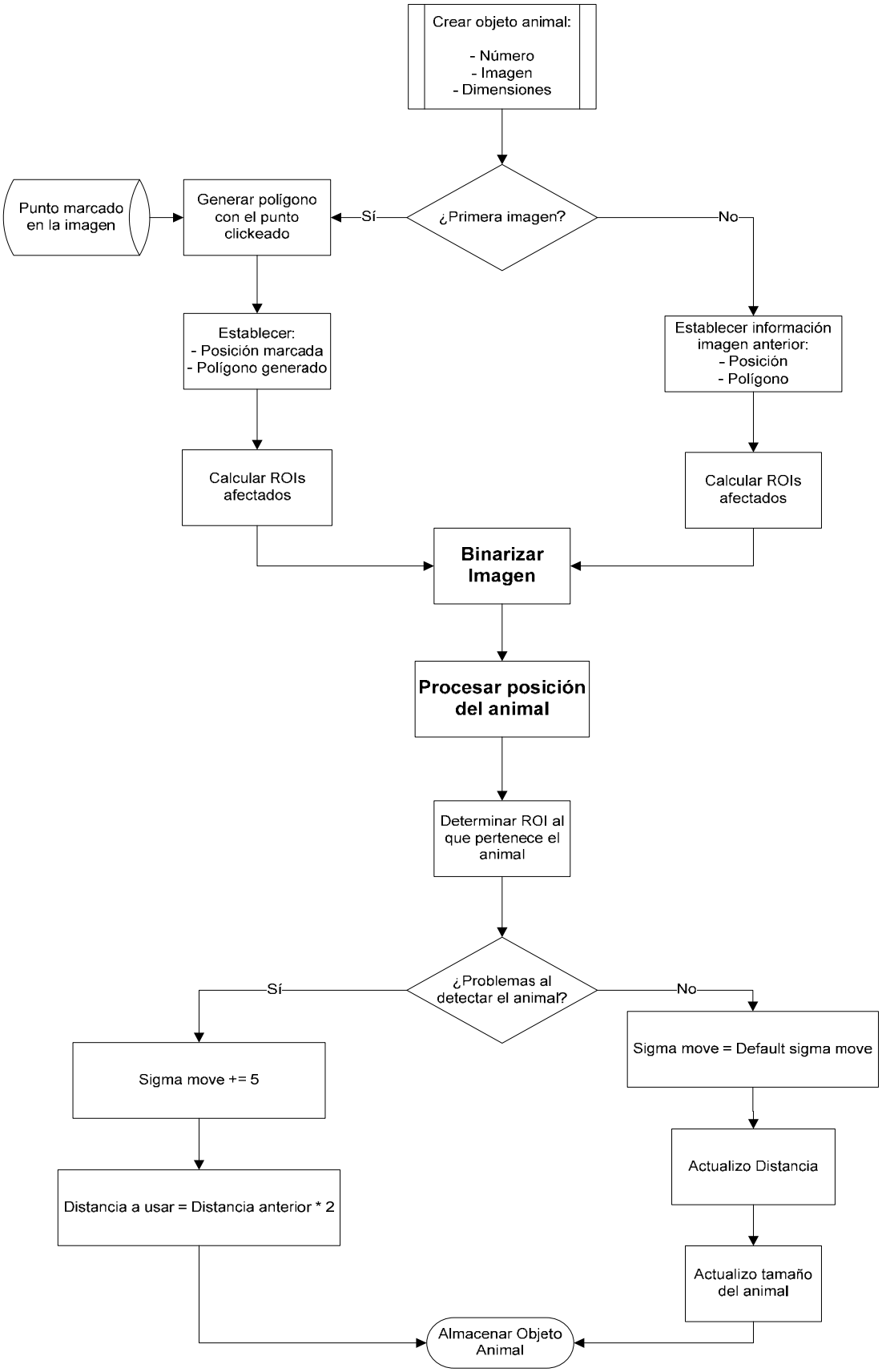
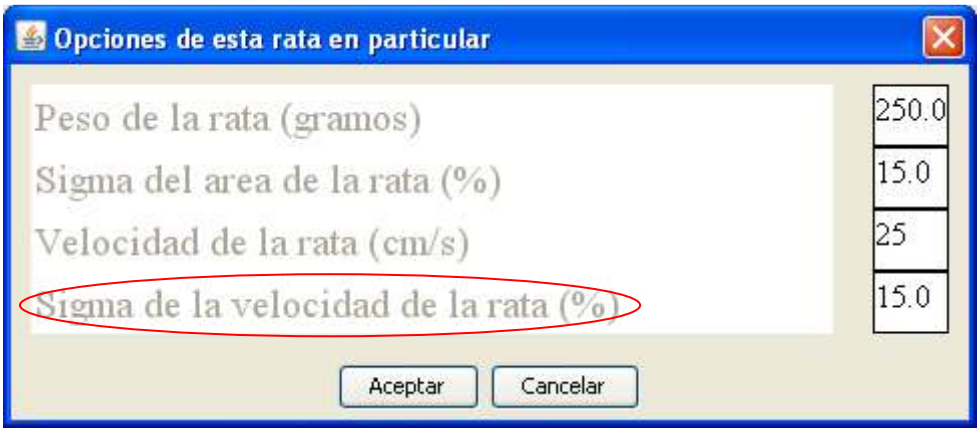


Diagrama de flujo del algoritmo detección del animal

Tras cambiar los parámetros, teóricamente se debe detectar el animal en el siguiente instante de tiempo. Tras hacerlo (en el siguiente instante o en el momento que se haga) es necesario restablecer los parámetros estadísticos a su valor original. Tanto la distancia como la varianza.

Por ello, cuando no se ha producido ningún error en la detección del animal en la imagen, se procede a establecer la varianza (sigma *Move*) a su valor por defecto (*default*), que coincide con el valor introducido por el usuario:



Opciones del animal

También se puede observar en las sentencias que se ejecutan cuando no se han producido errores en la detección dos procesos que nada tienen que ver con el tratamiento de los errores:

```
Actualizar distancia
Actualizar tamaño del animal
```

El porqué de estos procesos se explica en la siguiente página.

ACTUALIZAR PARÁMETROS DE DETECCIÓN

En este apartado se explicará cómo y por qué se cambian los parámetros de detección del animal durante el transcurso del experimento para adecuarlos al mismo.

Cuando el usuario introduce los parámetros característicos del animal que va a estudiar el programa transforma estos valores (velocidad en cm/s y peso en gramos) a valores relativos a la imagen (píxeles).

Aunque ya se ha explicado el procedimiento de transformación de estos valores para adecuarlos a las unidades que necesitamos, los píxeles, es importante recalcar que estos valores obtenidos muchas veces difieren de los valores reales que se obtendrán cuando se produzca el análisis digital de la imagen. Esta diferencia no se debe a la transformación únicamente, sino también a otros parámetros ajenos al programa.

Es posible que las condiciones de luz del experimento difieran con respecto a otro experimento, aunque sea el mismo animal el analizado. Esto puede conllevar a detectar un objeto mayor o menor en tamaño en los dos experimentos, sin embargo, según el peso, el tamaño debería ser el mismo.

Lo mismo ocurre con la velocidad del animal. Se puede establecer una velocidad por defecto que se supone que se moverá el animal. Pero es conocido el efecto de drogas como la cocaína, donde los animales experimentan una situación de euforia y éxtasis que se traduce en un aumento de la velocidad del mismo.

Por todo esto se pensó modificar los parámetros del algoritmo de detección, para que varíen y se adapten a los valores reales que se obtienen analizando la imagen.

Por ello se decidió añadir, en el caso de detección correcta del animal, unos procesos que actualizaran estos valores y permitiera aumentar la robustez del algoritmo frente a errores.

Esto también se puede interpretar como un sistema automático que consigue aprender de sus errores, es decir, se va adaptando a la realidad, aunque el usuario haya introducido erróneamente los valores.

Sin embargo, no es conveniente utilizar siempre el valor que se detectó en el instante anterior, sino que es más efectivo actualizar el valor que introdujo el usuario en el tiempo, y de manera gradual, hasta que conseguimos una adaptación perfecta.

Para ello, se realiza la siguiente operación en el ámbito del tamaño, que se implementa siguiendo la idea de un filtro:

$$tamaño_{teórico} = tamaño_{teórico} + \frac{tamaño_{real} - tamaño_{teórico}}{\alpha_{tamaño}}$$

De igual manera se actualiza la velocidad del animal:

$$velocidad_{teórica} = velocidad_{teórica} + \frac{velocidad_{real} - velocidad_{teórica}}{\alpha_{velocidad}}$$

En estos dos filtros se puede observar el uso de un alpha de tamaño y de velocidad. Este alpha es usado por el proceso para determinar la velocidad con la que el valor teórico se aproxima al valor real. Veamos un ejemplo:

$velocidad_{teórica} = 32$ velocidad real $= 26$ $\alpha_{velocidad} = 1.5$

$$Nueva\ velocidad_{teórica} = 32 + \frac{26 - 32}{1.3} = 32 + (-4) = 28$$

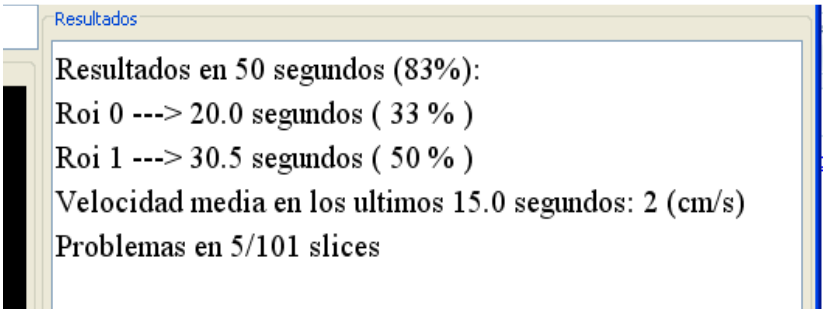
$velocidad_{teórica} = 32$ velocidad real $= 26$ $\alpha_{velocidad} = 2.5$

$$Nueva\ velocidad_{teórica} = 32 + \frac{26 - 32}{2.5} = 32 + (-2.4) = 29.6$$

Esto demuestra la necesidad de ajustar todos los parámetros que se utilizan en la detección para conseguir un algoritmo robusto, ya que hasta el último detalle es necesario para el correcto funcionamiento del proceso.

RE-ANÁLISIS DE LAS IMÁGENES PROBLEMÁTICAS

Hasta este momento, nuestro algoritmo ha analizado todas las imágenes generadas en el transcurso del experimento. Como podemos observar en la figura, conocemos el número de *slices* problemáticas, y tal y cómo se explicó anteriormente para cada imagen problemática se almacena la imagen de ese instante, del instante anterior y del posterior.



Diálogo de resultados de la interfaz principal

Este método analiza los *slices* de tres en tres en sentido inverso a como se han introducido en el vector de *slices* problemáticos. Para estas tres imágenes se ejecuta un proceso análogo al realizado anteriormente de detección de posición. Sin embargo este proceso difiere en varios aspectos, como por ejemplo actualizar los parámetros característicos del animal (no se realiza la actualización).

Por tanto, lo que realiza este algoritmo es: Calcular los ROIs afectados, binarizar la imagen y encontrar los objetos de la imagen. Posteriormente se elige el objeto según los parámetros estadísticos, que deben estar ajustados después de haber sido actualizados durante el transcurso del experimento.

Lo bueno de este sistema es que se conoce la posición del instante siguiente al problemático, por lo tanto no hace falta cambiar los parámetros estadísticos tal y como se hacía cuando ocurría un problema al elegir el objeto. De esta manera la detección es más fiable, ya que tiene dos posibilidades de detectar el objeto adecuado: usando el instante anterior o el posterior.

Tras esto se ejecutan las rutinas para establecer la posición del animal, ajustar la elipse, obtener la cabeza y cambiar el estado del *slice* de problemático a detectado.

Otro de los procesos que cambian en este re-análisis es el hecho de liberar la memoria eliminando las imágenes del *slice*. Ya que hay que usar este *slice* en el instante anterior si se da el caso de que éste también es problemático.

Tras esto se almacena la imagen final si está activada la opción de *Debug* por parte del usuario.

Finalmente, tras re-analizar todos los instantes problemáticos se actualizan los resultados en el interfaz principal.

ROIs AFECTADOS

Durante el análisis de imagen hemos usado varias veces un método que calculaba los ROIs afectados dependiendo del punto en el que se encuentra el animal. El algoritmo que calcula los habitáculos en los que se puede encontrar el animal es el siguiente:

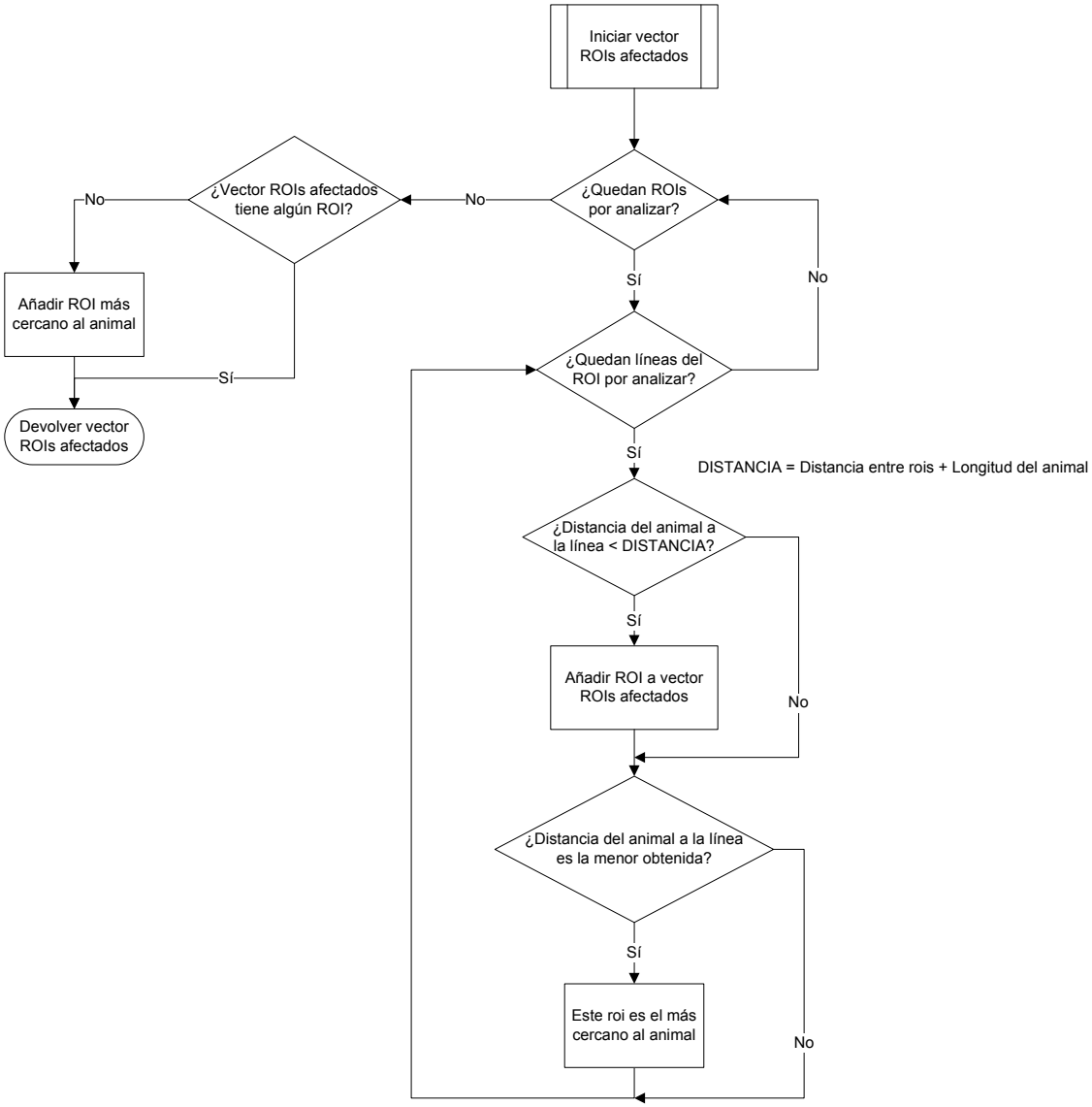


Diagrama de flujo para calcular los ROIs afectados

Lo primero que realizamos es inicializar el vector de ROIs afectados, donde guardaremos los polígonos (ROIs) donde se pueda encontrar el animal.

Posteriormente, para cada ROI dibujado ejecutamos las siguientes operaciones: Comprobamos la distancia de cada línea del ROI al animal. Si esta distancia es menor que **distancia_calculada**, añadimos el ROI al vector de ROIs afectados. Cuando esta distancia es la menor de todas las calculadas, se marca el ROI como el más cercano al animal.

Cuando hemos terminado de comprobar todos los ROIs, y si aún no se ha añadido ningún ROI al vector de afectados, se añade el ROI más cercano al animal. De esa manera nos aseguramos que siempre analizamos un ROI.

El único parámetro que nos queda por calcular es la **distancia_calculada**. Esta distancia debe ser tal que cuando el animal se acerque a la puerta se añada el ROI para análisis. Es decir, si es posible que el animal pueda estar en un ROI en el instante siguiente tenemos que añadirlo al vector de ROIs afectados.

Por tanto, esta **distancia_calculada** será la suma de la longitud del animal con la máxima distancia existente entre ROIs. Esto se debe a que hay casos en los que los habitáculos están separados por la puerta, y hay una cierta distancia entre ellos.

Para calcular la distancia entre ROIs ejecutamos el proceso descrito en el algoritmo de la siguiente página. En él calculamos la distancia existente entre el punto central de cada una de las líneas de un ROI con el punto central de las líneas del ROI a comparar. Se usa el punto central ya que en nuestros habitáculos la puerta está en el centro.

Tras comparar todas estas distancias, elegimos la mayor distancia entre ROIs, y ésta será la **distancia_calculada**.

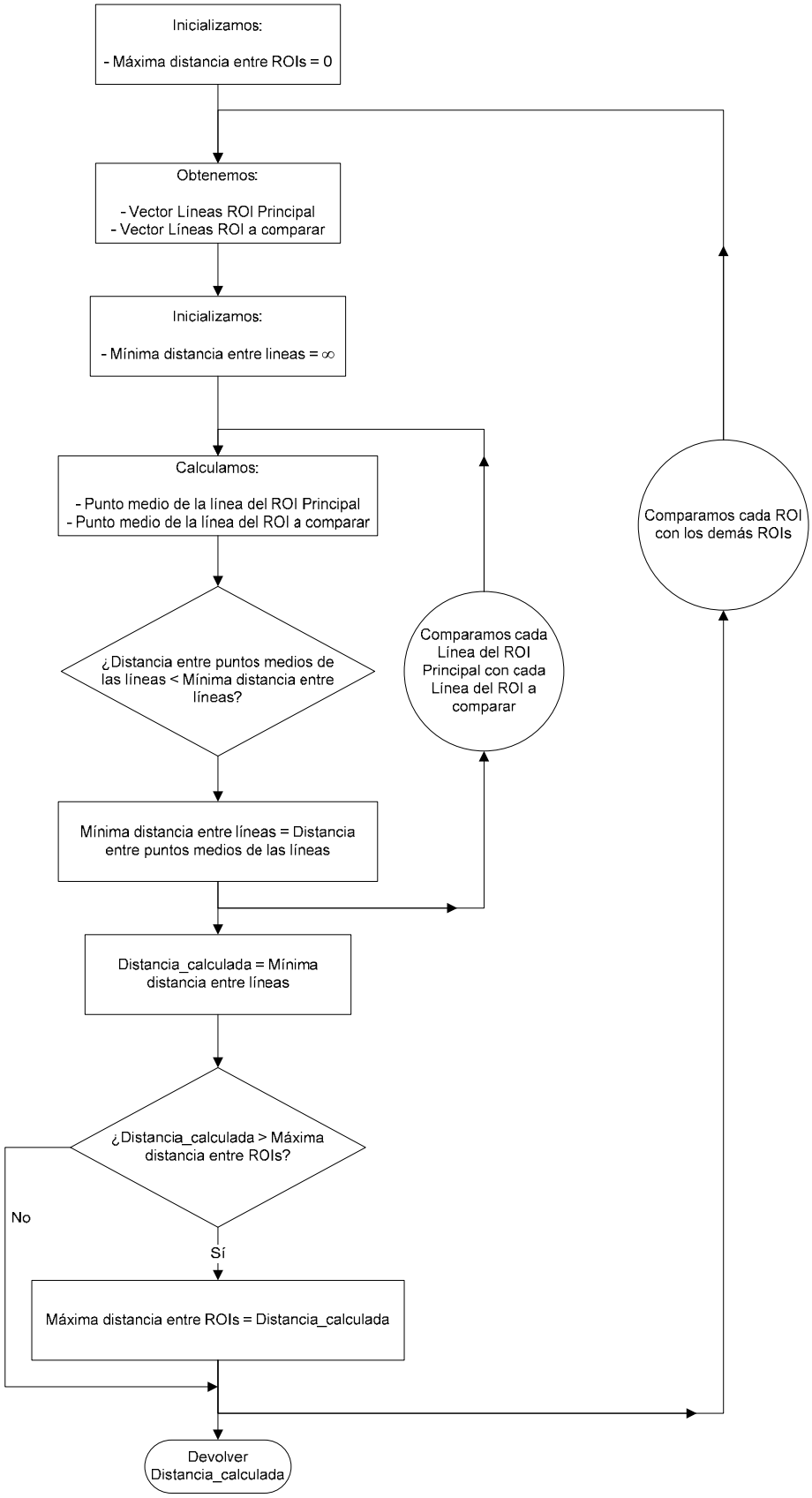


Diagrama de flujo del algoritmo para calcular la maxima distancia entre ROIs

ANÁLISIS DE RESULTADOS – FUTUROS OBJETIVOS

La siguiente funcionalidad que se quiere añadir al programa es un estudio de la trayectoria del animal. Con ello se quiere conseguir un análisis que refleje más información sobre el animal, para de esta manera poder comparar objetivamente más parámetros que el tiempo de estancia en cada cubículo.

Una manera de realizar esto sería catalogar el estado del animal en cada momento. Definir por ejemplo los estados de:

- Parado: la velocidad es menor que una establecida.
- En movimiento: La velocidad es mayor, y la distancia entre el punto inicial y el final es mayor que determinado valor.
- Movimiento aleatorio: el animal está en movimiento pero no sale de un cuadrante, está dando vueltas

También sería de utilidad dar más información sobre estos estados, ya que es diferente estar parado cerca de una pared, que en el centro de la sala, o entre dos habitáculos.

Es además de utilidad estudiar la torsión del animal a lo largo del tiempo, ya que puede ser interesante saber si el consumo de cocaína provoca el retorcimiento del animal. Además se podría analizar con este mismo fin el tiempo que pasa del total puesto en pie.

CONCLUSIONES

Este proyecto ha servido para desarrollar una herramienta de seguimiento de animales por video (*videotracking*) conjuntamente con el Departamento de Toxicología de la Universidad San Pablo – CEU. Esta herramienta software permite automatizar el proceso de estudio de los efectos reforzadores de las drogas de abuso mediante la técnica de condicionamiento espacial (CPP).

Esta herramienta permite a su vez mostrar los resultados de manera objetiva, de manera que se puede estudiar el comportamiento de varios animales y comparar en busca de patrones comunes en la conducta de éstos.

La herramienta se ha desarrollado diferenciando claramente la lógica de presentación de la lógica de programa, modularizando cada una de ellas. En esta memoria se ha explicado el uso de la interfaz (lógica de presentación) además de los diversos algoritmos que componen la lógica de programa.

El módulo de cálculo perteneciente a la lógica de programa hace un uso intensivo de técnicas de procesamiento digital de imagen, además de otras operaciones propias desarrolladas exclusivamente para este proyecto. El proceso para detectar el animal en primer lugar elimina el ruido en la imagen, además de acentuar la diferenciación entre el objeto y el fondo de la imagen. Posteriormente binariza la imagen procesada y localiza el objeto. En último lugar se contrasta con información de los instantes anteriores e información introducida por el usuario antes del experimento si el objeto encontrado es el deseado. Tras analizar todos los instantes de tiempo, se vuelven a analizar las imágenes problemáticas utilizando información de los instantes anteriores y posteriores al error en la detección.

Todo el proceso descrito se puede realizar en un video ya grabado, o si es necesario también se ha preparado la herramienta para trabajar en tiempo real, donde la herramienta nos indica si el tiempo de proceso de nuestro PC es suficiente o es necesario mejorarlo.

El algoritmo implementado se ha preparado de tal manera que permite el análisis de varios experimentos simultáneamente, disminuyendo de esta manera en gran medida el tiempo total de proceso, ya que recordemos que en el estudio de los efectos reforzadores de las drogas de abuso se suele experimentar con 15 animales, y en tiempos de estudio de 30 minutos cada animal.

Por último destacar la intencionalidad de desarrollar la aplicación de tal manera que se pueda ejecutar en equipos de bajo coste, tanto PCs de bajo coste como cámaras convencionales, gracias al intenso uso de procesamiento digital de la señal.

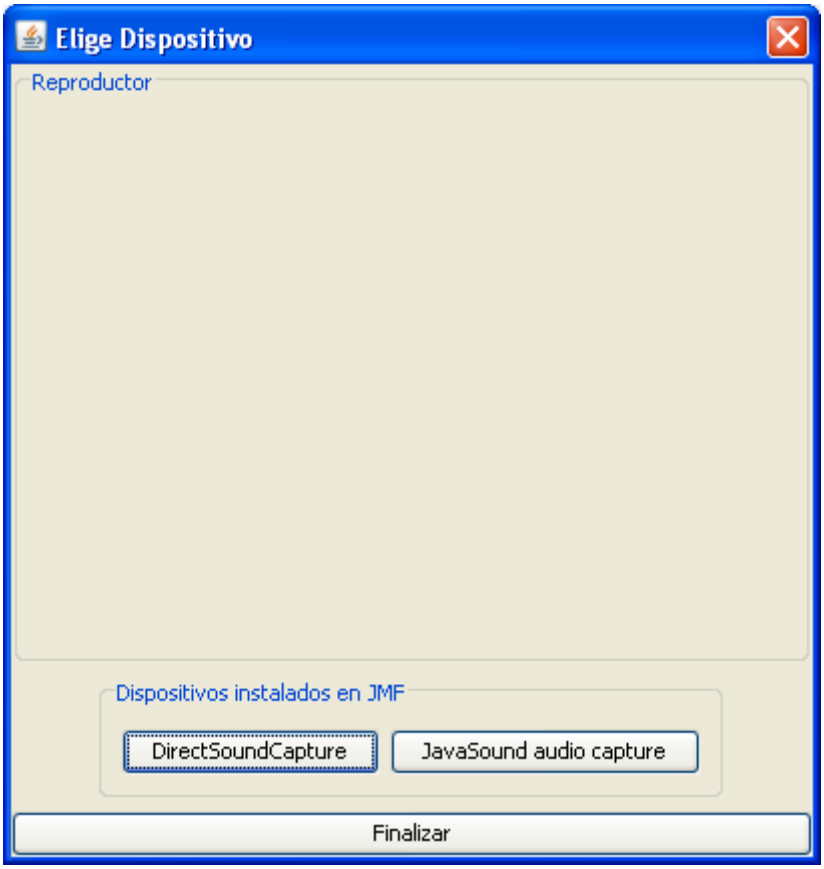
BIBLIOGRAFÍA

1. N. Otsu. A threshold selection method from gray level histograms. *{IEEE} Trans. Systems, Man and Cybernetics.* (1979).
2. Janusz Konrad. Motion Detection and Estimation. *Handbook of Image and Video Processing, páginas 207 - 225*. (2000).
3. P. Perner, Motion tracking of Animals for Behavior Analysis.
4. J. Morrow-Tesch, J. W. Dailey and H. Jiang. A Video Data Base System for Studying Animal Behavior. *Behavior Encyclopedia.*
5. Richard Vaughan, Neil Sumpter, Jane Henderson, Andy Frost and Stephen Cameron. Robot Control of Animal Flocks.
6. Alan J. Lipton, Hironobu Fujiyoshi and Raju S. Patil. Moving Target Classification and Tracking from Real-time Video.
7. Isaac Cohen and Gérard Medioni. Detecting and Tracking Moving Objects for Video Surveillance. *IEEE Proc. Computer Vision and Pattern Recognition.* (1999*).*
8. William Chen and Shih-Fu Chang. Motion Trajectory Matching of Video Objects.
9. Robert T. Collins, Alan J. Lipton and many more. A system for Video Surveillance and Monitoring. *VSAM Final Report.* (2000).
10. Giselle V. Kamenetzky, Alba E. Mustaca. Modelos animales para el estudio del alcoholismo. *Terapia psicológica*. (2005).
11. Fernández Tapia, A.M. Aportes del Darwinismo a la Psicología Clínica: El paradigma de la psicología evolucionista. *Terapia Psicológica* (2004).
12. Fidler, T., Bakner, L. & Cunningham C.L. Conditioned place aversión induced by intragastric administration of etanol in rats. *Pharmacology, Biochemistry and Behavior.* (2004).
13. Pavlov, I.P. Conditioned reflexes. An investigation of the physiological activity of the cerebral cortex. *Oxford: Oxford University Press.* (1927).
14. Bashir, Ashfaq A. Khokhar, Dan Schonfeld. Segmented trajectory based indexing and retrieval of video data. *Chicago*.
15. Helge Ewers, Alicia E. Smith, Ivo F. Sbalzarini, Hauke Lilie, Petros Koumoutsakos, Ari Helenius. Single-particle tracking of murine polyoma virus-like particles on live cells and artificial membranes. Zurich, Switzerland. (2005).
16. Birgit Möller, Stefan Posch. Analysis of object interactions in dynamic scenes. Halle/Salle.
17. Zhe Gu. Video database retrieval based on trajectory analysis. Canada. (1999).
18. Somboon Hongeng, Ram Nevatia, Francois Bremond. Video-based event recognition: Activity representation and probabilistic recognition methods. Los Angeles, California.
19. Ahmet Ekin, A. Murat Tekalp, Rajiv Mehrotra. Automatic soccer video analysis and summarization. New York.
20. Gérard Medioni, Ram Nevatia, Isaac Cohen. Event detection and analysis from video streams. Los Angeles, California.
21. Spanagel, R. Recent animal models of alcoholism. *Alcohol Research and Health*. (2000).

22. I. F. Sbalzarini, P. Koumoutsakos. Feature point tracking and trajectory analysis for video imaging in cell biology. Zürich, Switzerland. (2005).
23. Segmentación por umbralización. Método de Otsu. Quilmes. (2005).
24. M. T. Bardo, Rick A. Bevins. Conditioned place preference: what does it add to our preclinical understanding of drug reward?. Nebraska, Lincoln. (2000).
25. Luis Stinus. Los psicoestimulantes, farmacologia y dependencia a las drogas, de la molécula al comportamiento. Bordeaux.
26. Giselle V. Kamenetzky, Alba E. Mustaca. Modelos animales para el estudio del alcoholismo. Argentina. (2005).
27. Jordi Camí, John McGeary, Robert M. Swift. Drogadicción. Barcelona, España. (2005).
28. Addiction science network: Conditioned place preference method. (2000).
29. Donal J. Ecobichon, Ed. The Basis of Toxicity Testing. 2ª Edición- CRC Press, 1997

ANEXO 1 - AÑADIR DISPOSITIVOS A JMF

Como se explicó anteriormente, el programa implementado utiliza para la adquisición de imágenes el *API* de JMF. Si elegimos un dispositivo de captación de imágenes en tiempo real como por ejemplo una cámara, es necesario tenerla instalada en el equipo, y detectada por JMF. Tras ello aparecerá el dispositivo en el menú de elección de dispositivo:

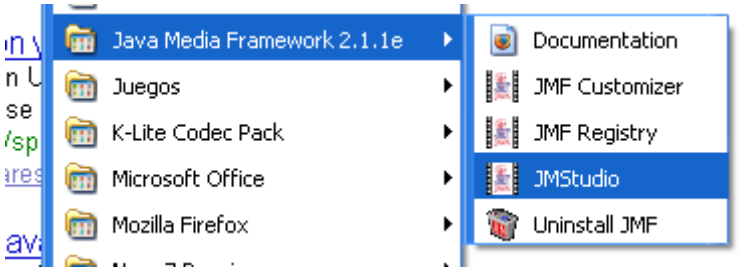


Elegir dispositivo

Por ello vamos a explicar en este anexo cómo se instala un dispositivo en JMF para poder usarlo en nuestro programa.

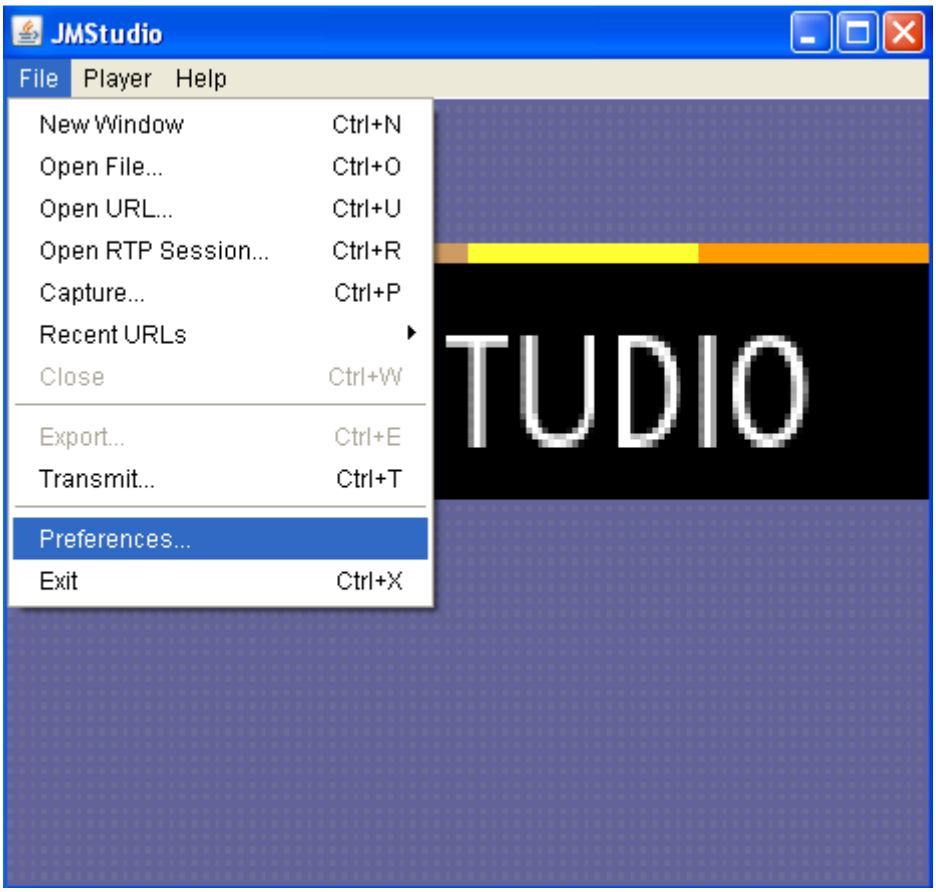
El primer paso es instalar los drivers correspondientes en el sistema, sin necesidad de instalar ningún otro programa.

Tras hacer esto abriremos el programa JMStudio, que se incluye en la instalación del paquete JMF: *jmf-2_1_1e-windows-i586*. Podemos observar el programa en la carpeta de inicio en la siguiente figura:



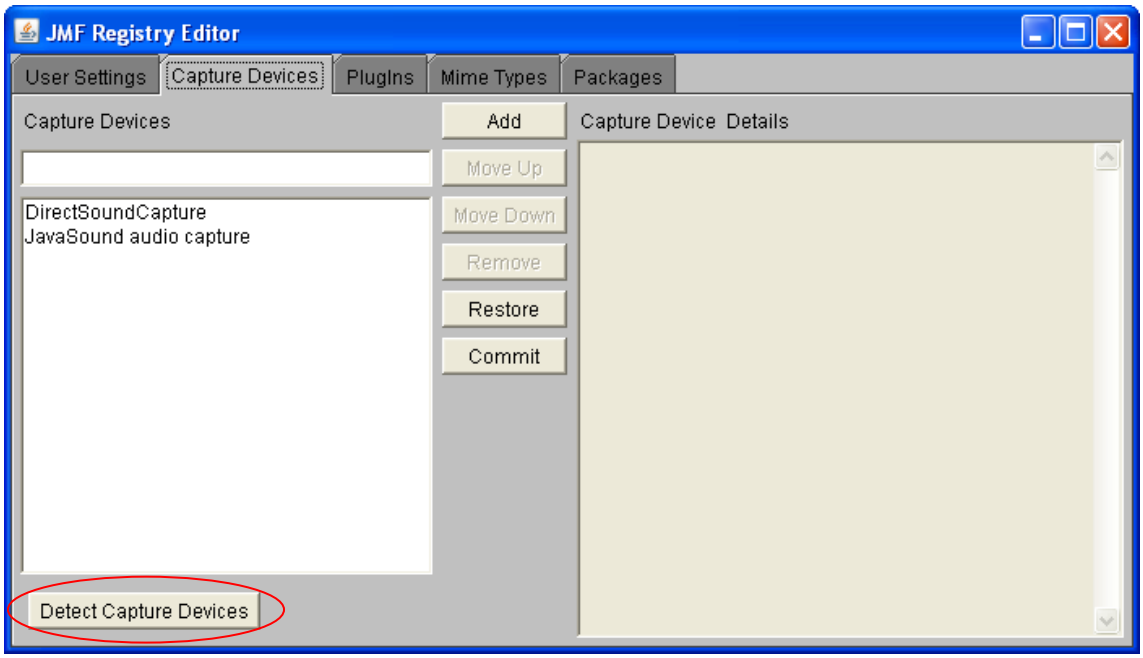
Ejecutar JMStudio

El programa JMStudio presenta la siguiente interfaz, de dónde marcaremos la opción preferencias, como se puede ver en la siguiente figura:



Interfaz JMStudio

En el menú de preferencias es necesario seleccionar la pestaña de *Capture Devices* (dispositivos de captura), y en ese menú seleccionaremos el botón de *Detect Capture Devices*. Tras hacer esto se añadirá el dispositivo al menú de dispositivos de captura.



Menú de preferencias

Una vez añadido el dispositivo podremos cerrar el programa JMStudio y empezar a utilizar nuestro programa.