Seguiment d'extremitats de ratolins en experiments optogenètics

Pere Parés Casellas

UOC - Màster en Enginyeria Informàtica

Tesi Final de Màster

David Masip Rodó

3 de gener de 2015

Índex

Índex de figures

Capítol 1

Introducció

Índex

1.1 Antecedents

El grup de recerca del *Neuroscience Institute* de la Universitat de Princeton, Nova Jersey, orienta la seva activitat d'investigació al camp de l'optogenètica, una tècnica utilitzada en l'àmbit de la neurociència que utilitza llum per a controlar neurones modificades genèticament per a esser-hi sensibles. Consisteix en una combinació de tècniques d'òptica i genètica per a controlar i monitorar les activitats de neurones individuals en teixit viu i mesurar de forma precisa els efectes d'aquestes manipulacions en temps real.

Aquest grup de recerca en concret duu a terme els experiments d'optogenètica en ratolins vius dins un entorn de proves controlat que els permet realitzar les manipulacions pertinents directament a les neurones i comprovar quin efecte tenen sobre l'aparell motor de l'animal.

Un tema que els és d'especial interès és solucionar la problemàtica de poder determinar en tot moment la posició de les potes del davant i del darrere a partir d'algorismes de visió artificial aplicats a vídeos capturats a l'entorn real de prova. En aquest punt és precisament on s'introdueix l'objecte de treball d'aquesta tesi final de màster.

1.2 Objectius

L'objectiu principal d'aquest treball és aplicar tècniques de visió artificial per aconseguir localitzar i fer el seguiment de les extremitats dels ratolins dins l'entorn de prova. Més concretament:

- Aconseguir segmentar el ratolí distingint-lo de la resta de l'entorn en un vídeo.
- Aconseguir localitzar les extremitats visibles de l'animal i dur-ne a terme el seguiment en un vídeo.
- Ser capaços d'extreure mesures valuoses a partir de la localització anterior.
- Ser capaços d'avaluar l'èxit de la detecció per cada fotograma dels vídeos proporcionats.
- Proporcionar un *software* que pugui ser d'utilitat com a eina per l'equip de recerca del *Neuroscience Institute*.

1.3 Característiques del problema

Per a atacar el problema acabat de descriure, es disposarà de vídeos capturats dins l'entorn controlat de proves des d'una perspectiva semblant a la que es pot veure a la Figura 1.1 amb certes característiques que es descriuran a continuació:

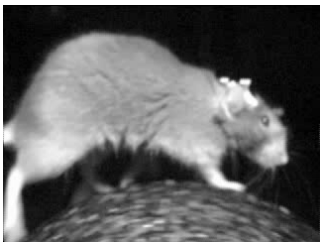


Figura 1.1: Captura dins l'entorn de prova

- L'entorn de proves consisteix en fixar el cap de l'animal i deixar la resta del cos lliure sobre una roda per permetre'n el moviment. L'entorn es manté a les fosques i es grava el comportament del rosegador amb una càmera i llum infraroja durant tot l'experiment.

- Els vídeos mostren el ratolí en moviment constant i ràpid. Tot i l'alt *frame rate* amb el que es capturen els vídeos, les extremitats apareixen borroses en alguns punts, dificultant-ne la segmentació.

- Les imatges proporcionades són de baixa resolució, característica necessària per a poder obtenir un *frame rate* de captura elevat amb la càmera de la que disposen.

- Com ja s'ha comentat, degut a que els experiments s'han de dur a terme a la foscor, les imatges es capturen utilitzant llum infraroja. Això comporta que existeixi poca informació de color.

- Degut a la naturalesa dels experiments, el cap del ratolí sempre es manté fixe. Això comporta que el cap sempre es trobi al mateix lloc respecte el primer *frame* del vídeo i és una informació valuosa que podria ser utilitzada durant el processat de l'imatge.

- El cos del ratolí presenta sempre un alt contrast amb el fons.

- Les extremitats del ratolí són molt flexibles, deformables i mai presenten una forma constant, sobretot quan es troba en moviment.

Tenint en compte totes aquestes característiques caldrà, doncs, dur a terme la segmentació i seguiment de les extremitats juntament amb la resta d'objectius descrits a l'apartat anterior.

1.4 Abast

La pretensió d'aquest projecte és desenvolupar un *software* que sigui capaç de dur a terme una segmentació i seguiment de les extremitats dels ratolins amb suficient robustesa i aportant informació suficientment valuosa com per a poder ser utilitzant posteriorment per part del grup de recerca del *Neuroscience Institute* com a utilitat auxiliar.

Així doncs, a part de proporcionar el *software* final també caldrà desenvolupar eines que permetin mesurar aquesta robustesa per poder elaborar les conclusions pertinents.

1.5 Elecció de la tecnologia

La forta vessant d'aquest projecte en l'àmbit de la visió per computador fan indispensable la utilització de llibreries de visió especialitzades per a desenvolupar els diversos algorismes que desembocaran a la solució final. Es van triar les llibreries *open source* i multi plataforma *OpenCV* (*Open Computer Vision v2.4.9*) per aquesta finalitat degut, en part, a l'experiència anterior de l'autor en aquest camp i l'ampli reconeixement d'aquestes en l'entorn professional i acadèmic (veure Secció 3.2). A més, aquestes llibreries desenvolupades en *C/C++* es troben molt ben integrades amb el llenguatge *Python* (veure Secció 3.1), l'escollit per a dur a terme aquest projecte.

Un dels punts crítics del projecte és que calia fer moltes proves de forma àgil sobre fotogrames de vídeos i, per aquest motiu, calia muntar un entorn de prova molt dinàmic. Un dels criteris per haver escollit el llenguatge de programació *Python* (*v2.7.3*) és que aquesta agilitat i dinamisme que es requereix lliga molt amb la filosofia darrere aquest llenguatge. Un altre criteri que es va tenir en compte a l'hora de prendre aquesta decisió és que té a la seva disposició altres llibreries molt útils que agilitzen el tractament d'imatges com *NumPy*, *SciPy* i *scikit-image* (veure els respectius apartats dedicats a aquestes llibreries dins la Secció 3.1). Per contra, un dels desavantatges d'utilitzar *Python* per a dur a terme tractaments d'imatge és que l'execució és molt més lenta que amb una implementació anàloga en *C* o *C++* (sobretot alguns algorismes que s'han implementat a mà, com els que es fan servir per a trobar l'esquelet topològic de l'animal).

Capítol 2

Planificació

En aquest apartat s'explica l'organització temporal necessària per a dur a terme tots els objectius del projecte pel que fa a l'estudi dels diversos algorismes de visió artificial que poden dur a la solució final, la codificació del propi *software*, la validació dels resultats, l'extracció de conclusions i la redacció de la memòria i de la preparació de la presentació final. L'organització temporal s'estructura al voltant de les diverses entregues repartides durant el quadrimestre, que actuen com a pilars de la planificació:

- Primera entrega (02/10/2014): proposta inicial (definició d'objectius, característiques del problema, propostes de solució i breu revisió de l'estat de l'art).
- Segona entrega (30/10/2014): estructura de la memòria final i primers resultats del treball.
- Tercera entrega (27/11/2014): primera versió de la memòria final i resultats del treball.
- Entrega final (04/01/2015): memòria final i presentació del treball en vídeo.

A la Figura 2.1 es presenta el diagrama de *Gantt* que il·lustra la planificació temporal teòrica. La línia verd fosca es desgranarà a continuació amb els diversos algorismes que es pretenen provar i implementar per intentar arribar a una solució amb uns resultats acceptables. Per a entendre millor el diagrama, s'ha especificat el codi de colors següent:

- Blau cel: recull de requeriments.
- Taronja: fites principals (entregues al *RAC*).
- Verd: gruix de l'anàlisi, disseny, implementació, validació i extracció de conclusions de la solució final.
- Verd fosc: període durant el qual s'estudiaran i implementaran els algorismes que poden portar a la solució final.
- Groc: desenvolupament de la documentació.

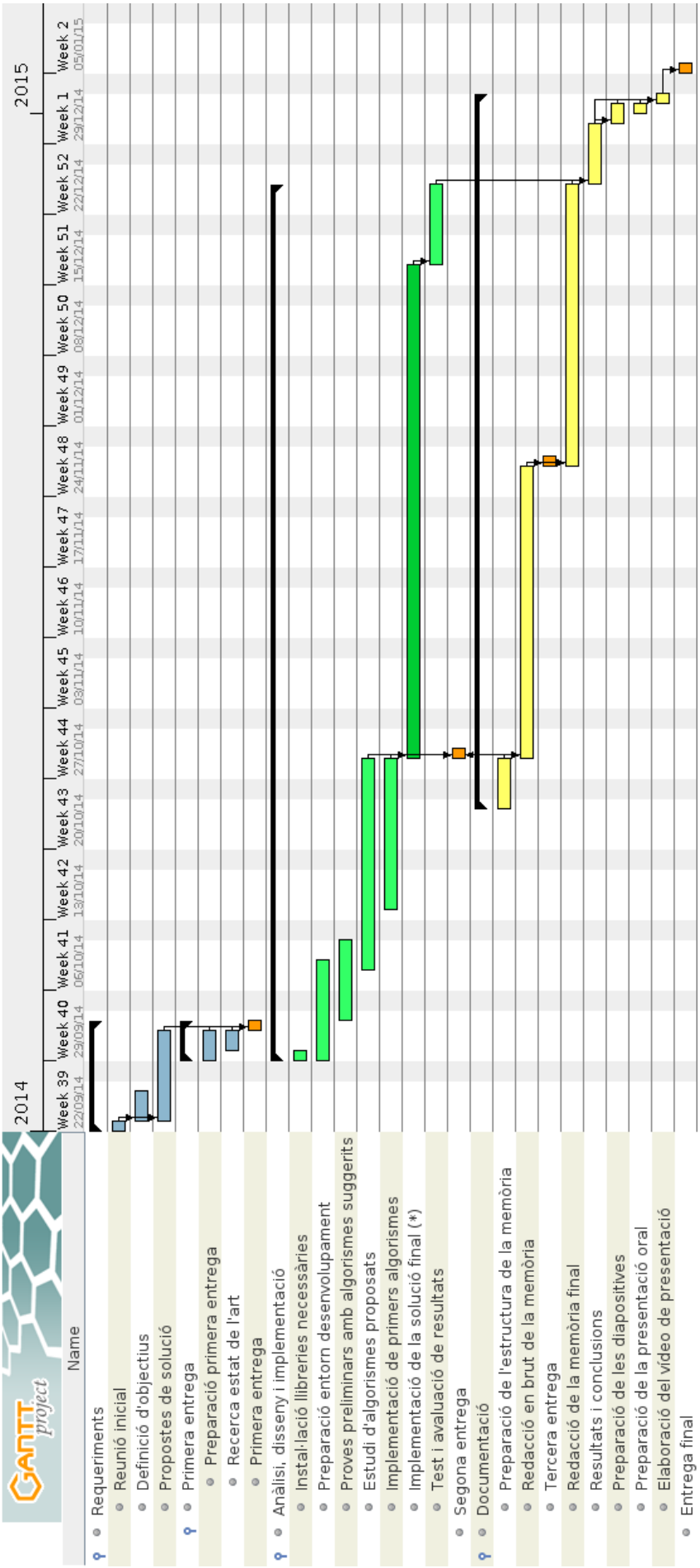


Figura 2.1: Diagrama de *Gantt*

Així doncs, vegem una explicació de les fites principals i un desglossament més detallat del període principal de desenvolupament:

1. Requeriments (22/09/2014 - 02/10/2014): durant aquesta primera fase marcada per la primera entrega, es va dur a terme la definició dels objectius del projecte així com la determinació de l'abast, una breu revisió de l'estat de l'art i les primeres propostes de solució.

2. Anàlisi, disseny i implementació (29/09/2014 - 23/12/2014): es tracta del període central del projecte i és on es duran a terme totes les activitats orientades a desenvolupar la solució final. Es pot dividir en els blocs més importants:

   - El primer pas consisteix en escollir la tecnologia a utilitzar per desenvolupar la solució final i la instal·lació de les llibreries i de l'entorn per començar a treballar (*OpenCV* 2.4.9 i *Python* 2.7.3).
   - Seguidament es planifica dur a terme unes proves preliminars amb alguns dels algorismes plantejats durant la fase de requeriments així com l'estudi d'alguns de més sofisticats.
   - Seguidament es planifica la implementació dels primers algorismes utilitzant el coneixement adquirit i, a partir d'aquí, decidir quins passos seguir dins el període que comprèn el gruix principal de desenvolupament del projecte.
   - El període més gran de desenvolupament s'ha representat amb una sola tasca verd fosca al diagrama de *Gantt* per dos motius: primer de tot, és degut a que l'espai disponible és limitat i, per altra banda, a que l'ordre en què s'implementaran i provaran els diversos algorismes és incert ja que dependrà dels resultats que es vagin obtenint. Les subtasques que comprenen aquest període són les següents:
     - Eliminació del fons utilitzant tècniques de *thresholding* amb diversos llindars o un d'adaptatiu.
     - Algorismes de segmentació:
       * *K-means* (segons els resultats obtinguts, considerar *C-means* o *EM*).
       * *Region growing*.
       * *Graph cutting*: considerar *GrabCut*, *GraphCut* i *MinCut*.
     - Trobar el cap de l'animal utilitzant *SIFT/SURF* o *Haar* i utilitzar aquesta informació per reforçar la segmentació.
     - Utilització de filtres de *Haar* entrenats amb extremitats en posicions diferents.

- L'últim tram d'aquest període es dedicarà a construir el que sigui necessari per a poder posar a prova els algorismes desenvolupats i validar els resultats obtinguts amb una quantitat suficient de vídeos de prova.

3. Documentació (24/10/2014 - 02/01/2015): El període que engloba l'elaboració de la documentació comprèn unes quantes fases relacionades. Primer de tot, es prepararà l'estructura general de la memòria. Seguidament, s'anirà realitzant la redacció en brut de la memòria durant el mateix període en el que s'anirà implementant la solució final. A partir d'aquesta redacció en brut es redactarà la memòria final i, finalment, s'afegiran els resultats obtinguts i les conclusions extretes de l'última fase de desenvolupament. Per acabar, es prepararan les diapositives i el vídeo de la presentació final.

Capítol 3

Conceptes Previs

Índex

En aquest capítol s'introdueixen alguns conceptes que requereixen una descripció més a fons al tractar-se d'eines que s'han hagut d'utilitzar per a trobar la solució a la problemàtica inicial.

3.1 *Python*

Python és un llenguatge de programació de propòsit general d'alt nivell amplament utilitzat[3]. La seva filosofia de disseny emfatitza la llegibilitat del codi i la seva sintaxi permet als programadors expressar conceptes en menys línies de codi de les que seria possible amb altres llenguatges com *C++* o *Java*.

Python dóna suport a múltiples paradigmes de programació (orientat a objectes, imperatiu, funcional, ...), presenta un sistema de *tipatge* dinàmic, una gestió automàtica de la memòria i una llibreria estàndard molt extensa.

Existeixen interpretadors de *Python* per la gran majoria de sistemes operatius, permetent que es pugui executar a gairebé tots els sistemes. A més, disposa d'eines de tercers que permeten empaquetar el codi en un executable independent sense haver de disposar d'un interpretador instal·lat.

El llenguatge de programació utilitzat per a desenvolupar el *software* d'aquesta tesi final de màster ha estat *Python* degut a motius ja esmentats però que es resumeixen a continuació:

- Les característiques del problema requereixen un entorn de desenvolupament àgil i dinàmic per a dur a terme proves el més ràpidament possible. Aquest llenguatge de programació aporta aquesta agilitat i dinamisme.
- Multi plataforma: no requereix una compilació diferent per cada sistema operatiu.
- Ben integrat dins les llibreries de visió artificial escollides, les *OpenCV*.
- Existència de llibreries molt útils pel que fa al tractament d'imatge: *NumPy*, *SciPy* i *scikit-image*

3.1.1 *Sphinx*

Sphinx és una eina per a documentar codi semblant a *Doxygen* o *Javadoc* molt estesa en el món *Python*. S'ha utilitzat notació *Sphinx* per a crear els *docstrings* del codi entregat com a solució final[6].

3.1.2 *Numpy*

NumPy és una extensió per al llenguatge de programació *Python*, afegint suport per a grans *arrays* multi dimensionals i matrius, juntament amb una llibreria per a dur a terme operacions sobre aquestes[1].

Utilitzant *NumPy* es poden expressar imatges com a *arrays* multi dimensionals de forma computacionalment eficient. De fet, la implementació d'*OpenCV* per a *Python* (*cv2*) utilitza *Numpy* internament.

3.1.3 *Scipy*

SciPy és una llibreria *Python* de codi obert utilitzat en l'àmbit de la computació científica[5]. Conté mòduls per a fer optimització, àlgebra lineal, integració, interpolació, transformades de *Fourier*, processat d'imatges i de senyals, entre altres.

Construeix la seva funcionalitat al voltant de *Numpy*.

3.1.4 *scikit-image*

scikit-image és una col·lecció d'algorismes d'alta qualitat i escrit per una comunitat activa de voluntaris per a dur a terme processat d'imatge[4] utilitzant, també, *Numpy* com a base.

Els algorismes inclosos són molt pròxims a l'estat de l'art en el camp de la visió per computador ja que s'integren molt ràpid directament de *papers* acadèmics.

3.2 *OpenCV*

OpenCV (*Open Source Computer Vision*) és una llibreria multi plataforma de funcions principalment orientada a la visió per computador en temps real[2], desenvolupat per *Intel Russia* i actualment suportat per Willow Garage. Està a disposició dels desenvolupadors sota la llicència de codi obert *BSD* i és gratuïta tant per usos comercials com acadèmics.

Disposa d'interfícies *C*, *C++*, *Java* i *Python* (amb suport per *Numpy*) i existeix per *Windows*, *GNU/Linux*, *Mac OS*, *iOS* i *Android*.

Es tracta del pilar tecnològic principal d'aquest projecte juntament amb el llenguatge de programació *Python* i es va escollir bàsicament per les raons següents:

- Multi plataforma.
- Experiència anterior amb la utilització d'aquestes llibreries.
- Ampli reconeixement en l'entorn professional i acadèmic.
- Bona integració amb *Python* i *NumPy*

Capítol 4

Solucions Algorísmiques

Índex

A continuació s'exposaran les diverses solucions algorísmiques que s'han provat i aplicat per assolir els objectius d'aquest treball. La solució final consisteix en la combinació de molts d'aquests algorismes, cada un d'ells solucionant porcions específiques del problema.

4.1 *Thresholding*

Un dels conjunts d'algorismes de visió més bàsics per a dur a terme una primera segmentació són els de *thresholding*. Tots ells consisteixen en obtenir una imatge binària a partir de cert llindar que, com es veurà més endavant, serviran per a identificar el cos principal de l'animal i, a partir d'aquí, aplicar algorismes més complexes per a obtenir una segmentació més bona.

Per a provar els diversos algorismes de *thresholding* disponibles es va desenvolupar un entorn que permetia anar modificant els valors dels diversos llindars i veure'n el resultat en temps real. Això va permetre arribar a conclusions més ràpidament i a poder triar el mètode de *thresholding* que es va considerar més adequat degudes les característiques del problema.

A continuació veurem les peculiaritats, avantatges i inconvenients de cadascun d'ells.

4.1.1 Binarització

El primer de tots, i el més simple, és el que realitza una binarització a partir de cert llindar de nivell de gris (Figura 4.1, llindar igual a 100). Consisteix en representar, de color blanc, tots els píxels el valor dels quals es trobin per sota del llindar escollit i, en negre, tots els que estiguin per sobre. Com es pot veure, tot i que el mètode és senzill, el ratolí queda molt ben segmentat i separat del fons. Algunes parts de la roda també queden segmentades però resultaria fàcil eliminar-les per àrea vigilant de no fer desaparèixer les extremitats en cas que quedin separades del cos principal. El problema d'aquest mètode és, però, trobar un llindar adequat i que aquest s'ha de definir manualment.

Figura 4.1: Binarització

4.1.2 Truncar

Un altre mètode de *thresholding* seria el de truncar el valor de cada píxel a partir de cert llindar. Això significa que no s'obté una imatge binària sinó que no es permet l'existència d'un nivell de gris més gran que el llindar. Tal i com es pot veure a la Figura 4.2, el ratolí queda gairebé tot representat pel mateix nivell de gris, havent posat el llindar a 100.

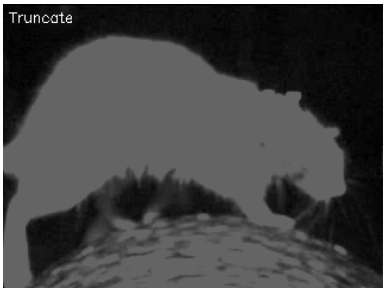


Figura 4.2: Truncar

4.1.3 A zero

Aquest mètode de *thresholding* consisteix en posar els píxels que es trobin per sota cert llindar a zero, completament negres. Això produeix imatges semblants al de la figura 4.3, on es pot veure el ratolí amb el seu color original i tot l'entorn completament negre. Aquest mètode pot servir per fer una segmentació del fons tot i conservant la resta d'informació del cos principal.

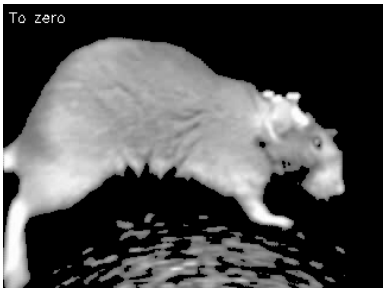


Figura 4.3: A zero

Un mètode basat en el mateix concepte que l'anterior però posant a zero els valors que es trobin per sobre de cert llindar i no per sota produiria resultats semblants al següent:

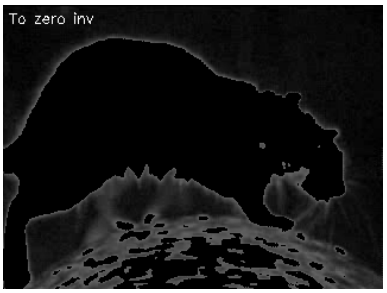


Figura 4.4: A zero invers

4.1.4 Llindars adaptatius

Tots els mètodes plantejats fins ara requereixen determinar un únic llindar a mà. Això pot ser un problema en imatges que presentin diferents condicions d'il·luminació en àrees diferents. Els mètodes adaptatius, per altra banda, intenten calcular un llindar per regions més petites de l'imatge, obtenint diversos *thresholds* per cada regió que permeten millors resultats en imatges on la il·luminació varia.

Un d'aquests mètodes adaptatius seria l'*Adaptive Mean*, que calcula el valor del llindar a partir la mitjana de l'àrea veïna. Tot i això, en el cas que ens ocupa, la diferència d'il·luminacions no és un problema ja que l'entorn de prova és molt controlat i, veient el resultat de la Figura 4.5, es pot comprovar que és massa sensible al soroll. Canviant la mida del bloc (veïns a partir dels quals es calcula el llindar) es pot millorar el resultat, obtenint menys *blobs*.

Figura 4.5: *Adaptive Mean*

Un altre mètode adaptatiu seria l'*Adaptive Gaussian*, l'idea del qual és la mateixa que la del mètode anterior però per calcular el llindar de cada regió no es fa servir la mitjana dels veïns sinó la suma ponderada dels veïns on els pesos són una finestra gaussiana. Amb aquest mètode no s'obtenen gaires bons resultats utilitzant les imatges amb les característiques del nostre problema tal i com es pot veure a l'imatge següent:

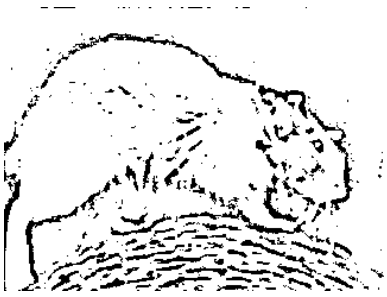


Figura 4.6: *Adaptive Gaussian*

4.1.5 Otsu

L'últim mètode de *thresholding* que s'ha provat, la binarització *Otsu*, assumeix que les imatges que tracta són bimodals. Això vol dir que assumeix que existeixen dos grans pics a la representació del nivell de gris de l'imatge en histograma per a calcular un llindar automàticament que els separi:

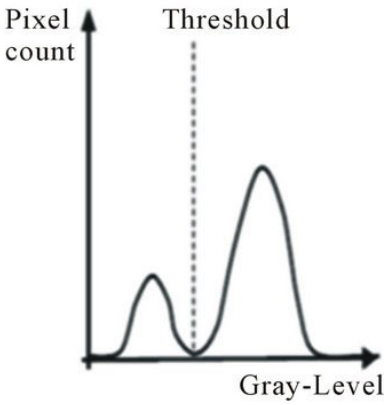


Figura 4.7: *Histograma bimodal*

En imatges no bimodals els resultats no són tan bons però en el nostre cas disposem d'imatges bimodals ja que una gran agrupació de píxels corresponen al fons i una altra gran agrupació corresponen al ratolí i parts de la roda. Com era d'esperar amb una imatge bimodal, els resultats de binarització són bons tal i com es pot veure a la Figura 4.8. L'eliminació dels petits *blobs* corresponents a parts de la roda és trivial utilitzant informació d'àrea però caldria anar amb compte ja que, en alguns casos, pot ser que les potes quedin separades del cos principal i s'eliminaria precisament la part que es vol localitzar.

Figura 4.8: *Otsu*

Per a eliminar soroll i obtenir contorns més suaus es pot passar un filtre gaussià per l'imatge abans d'utilitzar qualsevol binaritació, tal i com s'ha fet a la Figura 4.9. Amb uns quants *open/close* es podrien eliminar alguns forats petits i amb uns quants *erodes* i *dilates* consecutius es podrien eliminar alguns dels *blobs* més petits.

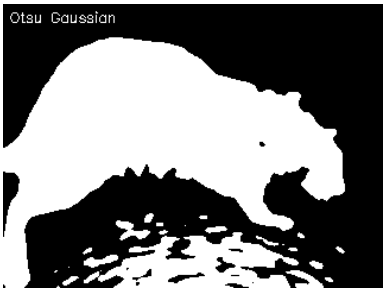


Figura 4.9: *Otsu Gaussian*

4.2 Contorns i gradient

Un altre tema que és d'especial interès és l'anàlisi del gradient i els contorns ja que poden ajudar a trobar les potes, considerant que es tracta de regions que canvien bruscament de nivell de gris.

Per a provar els diversos algorismes de cerca de contorns i gradient disponibles es va desenvolupar un entorn que permetia anar modificant els valors dels diversos llindars per veure'n els resultats en temps real.

4.2.1 Canny

Per a detectar vores, un dels algorismes més coneguts és el de *Canny*, considerat el més òptim degut a que intenta assolir les següents característiques:

- Bona detecció: l'algorisme intenta marcar tantes vores reals com sigui possible.

- Bona localització: les vores trobades s'intenten marcar el més pròximes possible a la vora de l'imatge real.
- Mínima resposta: certa vora de l'imatge s'hauria de marcar una sola vegada i, a ser possible, el soroll no hauria de crear vores falses.

A la Figura 4.10 es pot veure el resultat d'aplicar aquest algorisme a les imatges de test. Com es pot veure, a la bola rugosa es troben moltes vores però el contorn del ratolí es marca prou bé.

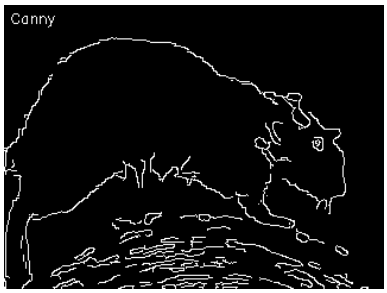


Figura 4.10: *Canny*

4.2.2 Otsu Laplacian

Un mètode que dóna bastants bons resultats és el resultant d'aplicar una binarització *Otsu* (que, com s'ha vist a la secció anterior, és el que dóna millors resultats) i, seguidament, aplicar un detector de vores Laplacià. Detectar vores a partir d'una imatge binària és fàcil i, com es pot veure, marca correctament tot el contorn del ratolí.

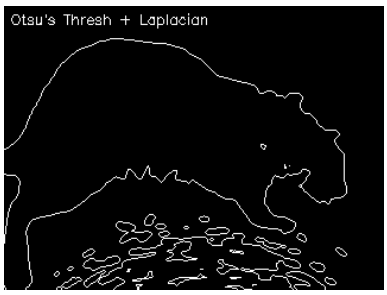


Figura 4.11: *Otsu Laplacian*

4.2.3 Sobel

Per a detectar el gradient en una imatge es pot calcular l'aproximació de la derivada d'aquesta utilitzant *Sobel*. El resultat obtingut a la Figura 4.12 és la combinació de la derivada en x i en y de l'imatge original amb un filtratge Gaussià previ per a eliminar soroll. Com es pot

veure, els contorns de les potes tenen molt de gradient i ens pot servir com a informació per a poder fer el *tracking* requerit.

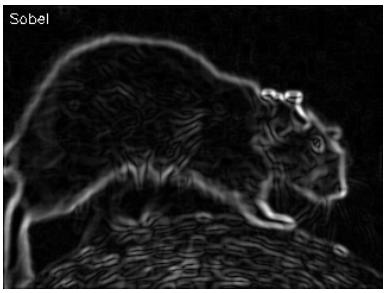


Figura 4.12: *Sobel*

4.2.4 *Scharr*

Una forma més precisa de calcular la derivada d'una imatge utilitzant una finestra de *3x3* seria utilitzant la funció *Scharr*. Com es pot veure a la Figura 4.13, però, és més sensible al soroll i marca molt més els contorns.

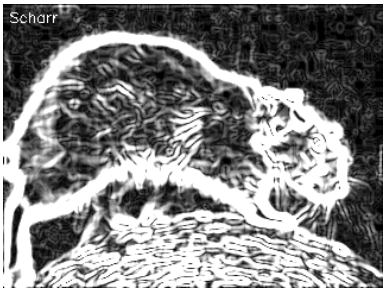


Figura 4.13: *Scharr*

4.3 *Esquelet topològic*

Uns altres algorismes que poden resultar interessants a l'hora de localitzar les extremitats de l'animal són els que permeten construir l'esquelet tipològic d'un objecte binaritzat. Aquests duen a terme operacions de morfologia matemàtica sobre l'imatge de tal manera que la van "aprimant" fins que només queda una versió fina de la forma original que és equidistant als contorns inicials.

Existeixen bastantes implementacions per a calcular esquelets topològics, i per aquest treball se n'han provat tres. Vegem com es comporten partint de l'imatge binària de la Figura 4.14

Figura 4.14: Imatge base binària

El primer algorisme implementat utilitzant les llibreries *Scipy* i operacions *erode* i *dilate* obté els resultats que es poden veure a la Figura 4.15; el codi font es pot trobar a l'Annex, Secció 2.2.2. Com es pot veure, els resultats són molt bons considerant que l'únic que s'ha fet prèviament ha estat binaritzar l'imatge. Com sempre, les rugositats de la roda on camina el ratolí porten problemes i caldrà aplicar aquests mètodes sobre imatges ja segmentades on aquesta ja no hi sigui tal i com es veurà més endavant.

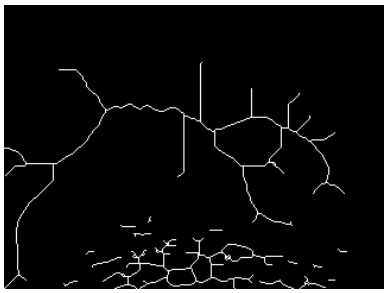


Figura 4.15: *Skeleton*

Un altre algorisme implementat es tracta del proposat per Zicheng Guo i Richard Hall [14], trobat fet en *C++*[1] i implementat en *Python* per a la realització d'aquest treball; el codi font es pot trobar a l'Annex, Secció 2.2.2. Els resultats obtinguts es poden veure a la Figura 4.16 i, com es pot apreciar, també són bastant bons.

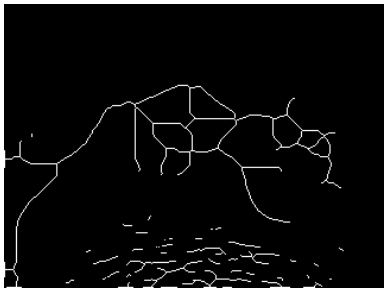


Figura 4.16: *Guo Hall*

[1] http://opencv-code.com/quick-tips/implementation-of-guo-hall-thinning-algorithm/

L'últim algorisme implementat per a calcular l'esquelet del ratolí és el proposat per T.Y. Zhang i C.Y. Suen [11], trobat fet en *C++*[2] i implementat en *Python* per a la realització d'aquest treball; el codi font es pot trobar a l'Annex, Secció 2.2.2. Els resultats obtinguts es poden veure a la Figura 4.17.

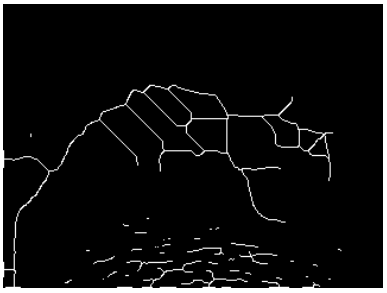


Figura 4.17: *Zhang Suen*

4.4 Segmentació

Un pas fonamental per assolir l'objectiu de localitzar les extremitats del ratolí en tots els fotogrames dels vídeos consisteix en obtenir una bona segmentació de l'animal. L'idea original en la planificació era provar bastants més mètodes dels que s'han acabat analitzant però al final no s'ha considerat convenient ja que *Grab-cut* ja proporciona resultats molt satisfactoris. A continuació s'exposen els mètodes provats.

4.4.1 K-means

K-means és un mètode de divisió de dades en k clústers que pot ser utilitzat en visió artificial per agrupar els píxels d'una imatge en k regions obtenint una imatge segmentada. El primer problema que presenta és que s'ha de triar prèviament el nombre de clústers que utilitzarà l'algorisme i això en determinarà el nombre de regions obtingudes. Un altre problema que presenta quan s'aplica en segmentació d'imatges és que no té en compte la distribució espacial dels píxels ja que només els agrupa per color i això provoca que s'obtinguin agrupacions de píxels separats entre si pertanyents a la mateixa regió.

Consisteix en definir aleatòriament uns punts inicials, els *centroides*, on cada píxel s'hi agruparà per distància. A cada iteració de l'algorisme, el *centroide* es mourà al punt mig dels píxels que hi han quedat agrupats i, un cop fet, cada píxel es tornarà a agrupar amb el *centroide* que l'hi hagi quedat més aprop. Això s'anirà repetint fins que no hi hagi més moviments de píxels o fins que hagi passat cert nombre d'iteracions.

[2] http://opencv-code.com/quick-tips/implementation-of-thinning-algorithm-in-opencv/

A continuació es mostraran alguns resultats obtinguts amb imatges de prova utilitzant diversos valors de k.

Amb dos clústers el resultat és molt semblant al de la binarització *Otsu* (Figura 4.8) degut a que les imatges capturades dins l'entorn de proves són bimodals. Els píxels del fons queden tots agrupats en un sol clúster i els del ratolí i algunes parts de la roda queden agrupats en un altre. El problema de mateixes regions distribuïdes en l'espai és evident veient aquest resultat.

Figura 4.18: *K-means*; k=2

Amb tres clústers s'obté una regió amb el fons, una altra amb el contorn del ratolí i algunes parts de la roda i l'últim amb l'interior del ratolí i algunes parts petites de la roda tal i com es pot veure a la Figura 4.19.

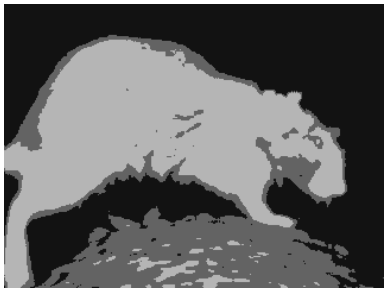


Figura 4.19: *K-means*; k=3

Amb quatre regions ja es comença a separar una mica l'anatomia del ratolí (zones del cap, morro, potes - Figura 4.20) en regions però no en clústers diferents.

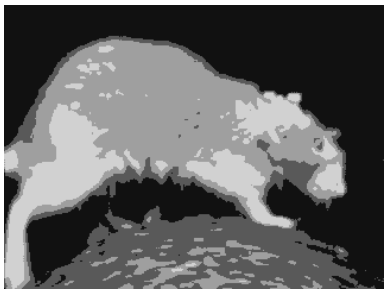


Figura 4.20: *K-means*; k=4

Una de les idees originals consistia en millorar la segmentació utilitzant, a més dels valors d'intensitat de cada píxel, altres valors de reforç com la textura per veure si es podien segmentar les potes soles. Això al final no s'ha dut a terme ja que aquest mètode de segmentació es va acabar descartant degut a que, primer, per dos clústers ja donava uns resultats semblants als obtinguts amb Otsu, segon, que per més clústers es trobaven regions sense continuïtat espacial degut a les característiques de l'algorisme i això és de poca utilitat en aquest cas, tercer, que el nombre de clústers s'ha de definir prèviament i, quart, que el mètode *Grab-cut* ja proporciona resultats molt bons per si sol.

4.4.2 Grab-cut

Es tracta d'un algorisme de segmentació d'imatge supervisat i interactiu basat en la poda de grafs. L'idea és aplicar aquest algorisme automatitzant la part supervisada amb altres mètodes per a que acabi essent un procés del tot automàtic, tal i com s'explicarà més endavant.

Grab-cut comença a partir d'un rectangle definit per l'usuari al voltant de l'objecte que cal segmentar. L'algorisme estima la distribució de color de l'objecte i el del fons utilitzant un model Gaussià. Això s'utilitza per a construir un camp de Markov aleatori sobre les etiquetes dels píxels, amb una funció d'energia que prefereix regions connectades que tinguin la mateixa etiqueta, i executant una optimització basada en la poda de grafs per a inferir els seus valors. Com que aquesta estimació és probablement millor que la recuperada del rectangle original, aquest procés es va repetint fins que s'arriba a convergir. Un cop es disposa d'aquesta estimació, l'usuari pot corregir el resultat marcant regions mal classificades com a fons o com a objecte.

Vegem els passos que segueix amb una imatge de prova:

1. Primer de tot, l'usuari ha de marcar un rectangle al voltant de l'objecte a segmentar:

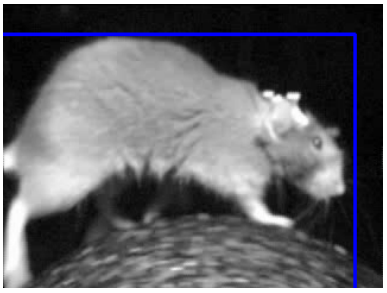


Figura 4.21: *Grab-cut* - rectangle manual inicial

2. S'aplica el primer pas de *Grab-cut* sobre l'imatge marcada i s'obté el següent resultat:

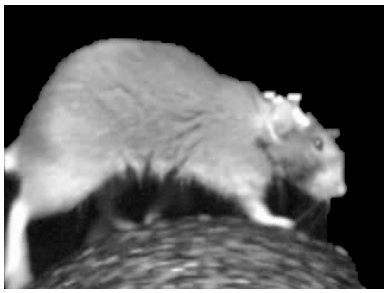


Figura 4.22: *Grab-cut* - resultat amb el rectangle manual inicial

Es pot veure com la part superior de l'animal ja queda ben segmentada però la part inferior es confon amb la roda i amb el fons entre les cames.

3. Un cop fet el primer pas, el mètode retorna una màscara amb els píxels marcats com a possible *background* i possible *foreground*. El que es pot fer és modificar aquesta màscara marcant regions com a *background* segur o com a *foreground* segur per millorar el resultat. Així doncs, marquem manualment alguns píxels de la roda com a *background* segur:

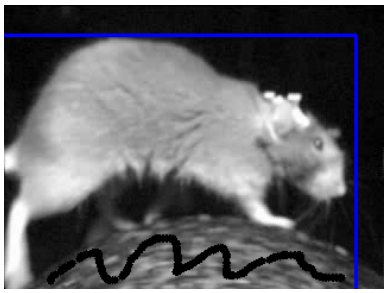


Figura 4.23: *Grab-cut* - regions de *background* marcades manualment

4. Amb aquesta nova informació es duen a terme algunes iteracions més i la bola i el fons entre les cames desapareixen, deixant el ratolí perfectament segmentat:

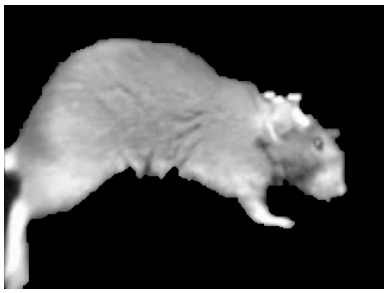


Figura 4.24: *Grab-cut* - Resultat manual final

Com es pot veure, aquest mètode obté resultats molt bons però, com s'ha vist, requereix

que l'usuari intervingui. Per a solucionar aquesta carència, s'ha automatitzat el procés utilitzant altres tècniques de visió artificial; vegem-ne el procediment:

1. Primer de tot, per obtenir el rectangle inicial de forma automàtica, s'ha dut a terme el següent:

   (a) Binarització *Otsu* (Secció 4.1.5) havent aplicat primer un filtre Gaussià per eliminar soroll.

   (b) Com que aplicant una binarització sempre queden parts de la roda, només ens quedem amb la regió amb àrea més gran.

   (c) A partir de la binarització on s'han eliminat els *blobs* més petits, es calcula el *bounding box* de l'objecte resultant. Aquest serà el que s'utilitzarà com a rectangle inicial que requereix *Grab-cut*.

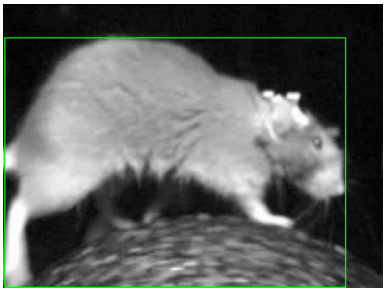


Figura 4.25: *Grab-cut* - Rectangle inicial obtingut automàticament

2. Seguidament, amb el rectangle resultant, s'executa el primer pas de *Grab-cut* obtenint resultats molt semblants als de la Figura 4.22.

3. Un cop executat el primer pas, s'obté una màscara amb etiquetes (regions marcades com a possible *background* i regions marcades com a possible *foreground*). Cal millorar aquesta màscara per obtenir resultats més bons:

   (a) Primer de tot es marquen els píxels del *blob* amb àrea més gran calculat abans com a possible *foreground*. Es marquen com a possibles ja que, a vegades, el *blob* més gran conté píxels que no són del ratolí (com parts de la roda que han quedat enganxades):

Figura 4.26: *Grab-cut* - *Blob* més gran com a *background* possible

Cal dir que en aquest cas el *blob* més gran ja és una segmentació molt bona però no sempre és així en tots els casos, o no faria falta aplicar mètodes de segmentació més complexos com aquest.

(b) Seguidament es marquen parts de la roda com a *backround* segur. Com que sabem que la bola sempre es troba a la mateixa posició, es dibuixa un cercle que n'enganxi bona part dels píxels (el centre i el radi es mantenen en un fitxer de configuració a part per si la bola canviés de lloc o de mida):

Figura 4.27: *Grab-cut* - Cercle inferior com a *background* segur

(c) Ajuntant aquesta informació amb la màscara obtinguda del primer pas de segmentació s'obté la màscara següent:

Figura 4.28: *Grab-cut* - Màscara resultant del procés automàtic

On el color negre és el que es troba marcat com a *background* segur, el color gris fosc està marcat com a *background* possible i el gris clar com a *foreground* possible.

(d) Amb aquesta nova màscara es torna a aplicar *Grab-cut* i el resultat és el mateix que l'obtingut de forma manual:

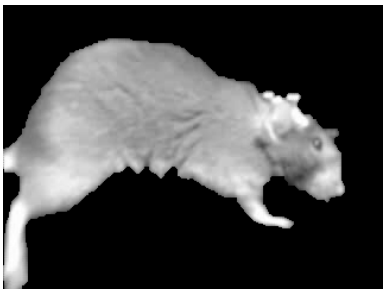


Figura 4.29: *Grab-cut* - Resultat final automàtic

Aquest algorisme ara sí que es pot aplicar de forma automàtica sobre tots els fotogrames dels vídeos sense interacció per part de l'usuari.

4.5 Classificació utilitzant característiques *Haar-like*

Les característiques *Haar-like* són característiques d'imatge digital utilitzades en reconeixement d'objectes a partir de moltes mostres positives i negatives. Deuen el seu nom a la similitud intuïtiva amb els *Haar wavelets* i van ser utilitzats en el primer detector de cares en temps real[12].

Cada una d'aquestes característiques considera regions adjacents rectangulars dins una posició específica d'una imatge en una finestra de detecció, suma les intensitats dels píxels a cada regió i calcula la diferència entre aquestes sumes. Aquesta diferència s'utilitza posteriorment per a categoritzar subseccions d'una imatge.

Durant la fase de detecció, es va movent una finestra per sobre l'imatge d'entrada i es calcula la característica *Haar-like* per cada subsecció. Aquesta diferència es compara amb el llindar après que separa els "no objectes" dels "objectes". Es requereix un gran nombre de característiques per a descriure un objecte amb suficient precisió ja que cada una d'elles és només un classificador *dèbil* i cal aplicar-les en cascada per a formar un classificador *fort*.

Les llibreries *OpenCV* proporcionen un entrenador i un detector; a continuació es descriuran els passos que s'han seguit per a entrenar un classificador de potes i un altre de caps de ratolí per a fer-ne la detecció als vídeos de prova.

4.5.1 Preparació de les dades d'entrenament

El primer pas per a l'entrenament del classificador consisteix en recollir moltes mostres negatives (imatges on no apareix l'objecte que cal detectar) i moltes mostres positives (imatges on apareix l'objecte que cal detectar). A més, cal crear un arxiu de text amb informació sobre aquestes imatges per l'aplicació d'entrenament d'*OpenCV* (tant per les negatives com per les positives), tal i com es descriurà en els apartats següents.

Per a facilitar aquesta feina manual s'ha creat el programa *haar_sampler.py* en *Python* que permet obrir un arxiu de vídeo i anar retallant regions rectangulars amb el ratolí, guardar-les com a mostres negatives o positives i anar avançant pels *frames* del vídeo per anar capturant mostres a mesura que es van creant, alhora, els arxius de text auxiliars pertinents. El codi font es pot trobar a l'Annex, Secció 3. El programa es crida de la següent manera des de línia de comandes:

```
python haar_sampler.py -v <videofile> -p <storing_path> -t <true_identifier> -f <
    false_identifier>
```

- *-v*: permet especificar el vídeo des d'on es volen extreure les mostres.
- *-p*: permet especificar on es guardaran les mostres capturades. Dins aquesta ruta es crearan dos directoris, 'positives' i 'negatives', per guardar les imatges positives i negatives respectivament amb noms consecutius:
  - *identificador_numèric.png*
- *-t*: permet especificar el primer identificador d'imatges positives per on es continuarà.
- *-f*: permet especificar el primer identificador d'imatges negatives per on es continuarà.

Un cop en execució, les següents comandes permeten controlar el comportament del programa:

- *'Esc'*: finalitza el programa.
- Ratolí: permet dibuixar rectangles que seran guardats com a mostres negatives o positives.
- *'t'*: al prémer aquesta tecla, les mostres es consideraran positives.
- *'f'*: al prémer aquesta tecla, les mostres es consideraran negatives.

- *'s'*: guarda la selecció actual com a positiva o negativa, segons el mode seleccionat amb *'t'* o *'f'*.
- *'n'*: avança al següent *frame* del vídeo.

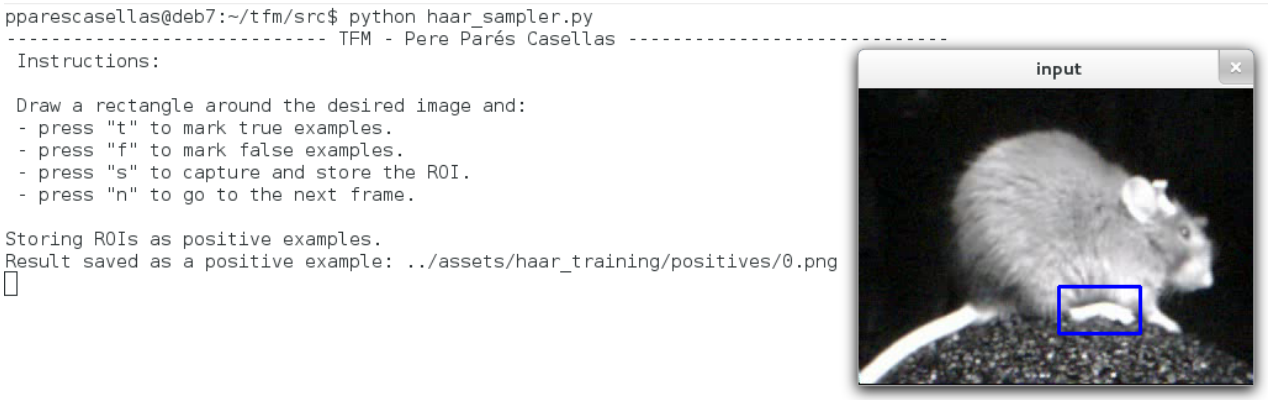


Figura 4.30: *Haar sampler* - exemple del programa creat per capturar mostres

Mostres negatives

Les mostres negatives han d'estar enumerades en un fitxer amb un format específic. Bàsicament es tracta d'un fitxer de text on s'informa de la localització de totes les imatges negatives. Considerant una estructura de directoris semblant a la següent:

```
/img
   0.png
   1.png
   ...
   n.png
bg.txt
```

El fitxer *bg.txt* hauria de contenir la següent informació:

```
img/0.png
img/1.png
...
img/n.png
```

Aquest fitxer es va creant automàticament a mesura que es van agafant mesures negatives utilitzant el programa *haar_sampler.py* ja mencionat.

Mostres positives

Les mostres positives cal generar-les utilitzant el programa *opencv_createsamples* a partir d'una única imatge o a partir d'una col·lecció d'imatges prèviament marcades. Per a entrenar el classificador s'ha optat per la segona opció: cal generar un arxiu de text semblant al que

descriu les mostres negatives però indicant la posició on es troba l'objecte a detectar dins l'imatge (nom de l'arxiu, nombre d'objectes, coordenades del *ROI* - x, y, amplada, alçada). Així doncs, seguint una estructura de directoris semblant a l'anterior, un exemple seria el següent:

```
positives/1.png  1  140 100 45 45
...
positives/n.png  1  100 200 50 50
```

Aquest fitxer també es va creant automàticament a mesura que es van agafant mesures positives utilitzant el programa *haar_sampler.py* ja mencionat.

Un cop es disposa de les mostres i del fitxer que les descriu, es pot utilitzar la comanda següent per a generar l'arxiu que servirà per entrenar el classificador:

```
opencv_createsamples -vec positives.vec -info info.dat
```

- *-vec*: nom de l'arxiu de sortida que contindrà les mostres positives per l'entrenament.
- *-info*: nom de l'arxiu que descriu la col·lecció d'imatges marcades.

4.5.2 Entrenament en cascada

El següent pas és l'entrenament del classificador utilitzant les mostres negatives i l'arxiu **.vec* amb les mostres positives. Com ja s'ha comentat, això es duu a terme utilitzant l'eina *opencv_traincascade*:

```
opencv_traincascade -data . -vec positives.vec -bg bg.txt -numPos 1000 -numNeg 2000
```

- *-data*: ruta on es generaran els arxius d'entrenament.
- *-vec*: arxiu que conté les mostres positives generat a partir de *opencv_createsamples* (Secció 4.5.1).
- *-bg*: arxiu que descriu les mostres negatives (Secció 4.5.1).
- *-numPos*: nombre de mostres positives.
- *-numNeg*: nombre de mostres negatives.

Aquest procediment d'entrenament pot durar hores i, fins i tot, dies sencers en acabar. En el cas de l'entrenament per trobar les potes del ratolí, es van utilitzar els següents paràmetres:

```
PARAMETERS:
cascadeDirName: .
vecFileName: positives.vec
bgFileName: bg.txt
numPos: 1000
numNeg: 2000
numStages: 20
precalcValBufSize[Mb] : 256
precalcIdxBufSize[Mb] : 256
stageType: BOOST
featureType: HAAR
sampleWidth: 24
sampleHeight: 24
boostType: GAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: ALL
```

4.5.3 Classificació

Un cop finalitzat el procediment d'entrenament s'obté el fitxer *cascade.xml*, que és el que descriu el classificador en cascada. Utilitzant el classificador de potes entrenat específicament per aquesta tesi de màster s'obtenen resultats com el següent:

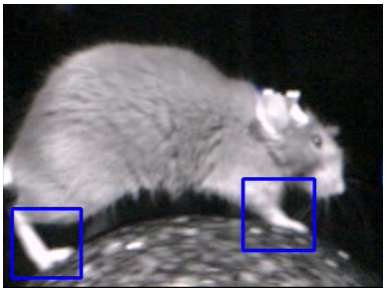


Figura 4.31: *Haar* - potes detectades utilitzant el classificador en cascada entrenat

Capítol 5

Disseny

En aquest apartat s'explicarà l'arquitectura *software* que s'ha dissenyat per a solucionar el problema del *tracking* d'extremitats. L'idea inicial era que el programari desenvolupat fos suficientment flexible i dinàmic com per dur a terme moltes proves de forma fàcil i que permetés aplicar els diversos algorismes de visió artificial en ordres específics per obtenir el processat final de cada fotograma. Per aquesta fi s'han aplicat alguns coneixements d'enginyeria del *software*.

Així doncs, es va pensar en encapsular algorismes de visió artificial en unitats de procés independents que permetessin rebre un *input* (una imatge, en aquest cas), processar-lo, enviar-lo al següent algorisme de la cadena i llavors en retornessin el resultat. Per a fer això es va pensar en el patró de disseny *chain of responsibility*:

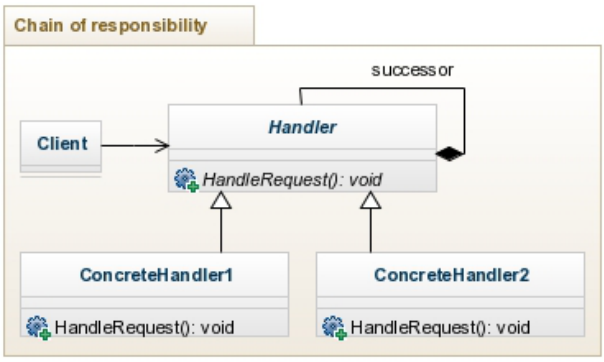


Figura 5.1: Patró de disseny *chain of responsibility*

Aquest patró permet implementar tants objectes *Handler* concrets com es vulgui, instanciar-los i lligar-los entre si formant una cadena de processat, just el que es necessita per aplicar els diversos algorismes de visió artificial successivament sobre cada fotograma:

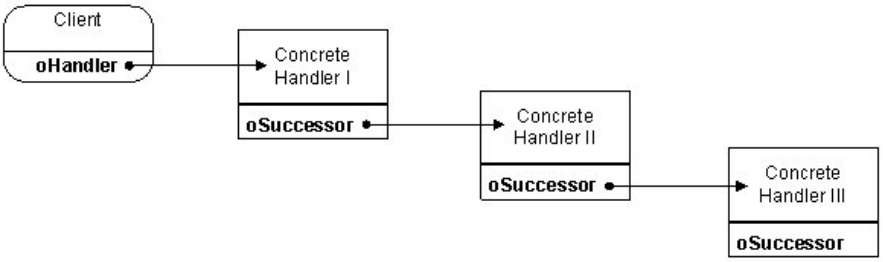


Figura 5.2: Cadena de processat

A més, la utilització d'aquest patró des d'estadis inicials del projecte va facilitar el fet d'anar aplicant múltiples processats a la mateixa imatge original, anant-ne variant l'ordre per fer proves, etc., aportant la flexibilitat i rapidesa requerides per l'entorn de prova.

Un altre aspecte important consisteix en mantenir desacoblat el programa de paràmetres estàtics. Per aquesta fi s'ha desenvolupat una classe que permet llegir un fitxer de configuració amb paràmetres variables fàcilment extensible, posant-los a disposició de tota la lògica del programa. Amb tots aquests conceptes en ment, es va dissenyar l'arquitectura *software* de la Figura 5.3. Vegem la descripció dels seus elements principals:

- *TFM* (*tfm.py*): es tracta de la classe principal, la que instancia tots els objectes, crea la cadena de responsabilitat i inicia el processat de tot el vídeo. El codi font es pot consultar a l'Annex, Secció 2.1. Per a una descripció més acurada, consultar els propis *docstring* dins el codi en format *Sphinx*.

- *Configuration* (*configuration.py*): conté tots els paràmetres de configuració necessaris pel bon funcionament de l'aplicació i els posa a disposició de tots els objectes que en requereixen. El codi font es pot consultar a l'Annex, Secció 2.3. Per a una descripció més acurada, consultar els propis *docstring* dins el codi en format *Sphinx*.

- *Processing* (*processing.py*): implementació del patró *chain of responsibility*; es tracta de la classe base de tots els objectes que contenen algorismes de visió artificial encapsulats susceptibles a ser encadenats. El codi font es pot consultar a l'Annex, Secció 2.2. Per a una descripció més acurada, consultar els propis *docstring* dins el codi en format *Sphinx*.

- *Segmentation* (*processing.py*): classe que encapsula tot el procés de segmentació automàtica del ratolí, explicat a la Secció 4.4.2. El codi font es pot consultar a l'Annex, Secció 2.2. Utilitza les funcions definides dins l'arxiu *segmentation.py*, veure l'Annex, Secció 2.2.1. Per a una descripció més acurada, consultar els propis *docstring* dins el codi en format *Sphinx*.

- *Skeleton* (*processing.py*): classe que encapsula tot el procés de cerca de l'esquelet topològic del ratolí, explicat a la Secció 4.3. El codi font es pot consultar a l'Annex, Secció 2.2. Utilitza les funcions definides dins l'arxiu *skeletonization.py*, veure l'Annex, Secció 2.2.2. Per a una descripció més acurada, consultar els propis *docstring* dins el codi en format *Sphinx*.

- *HaarHeadClassification* (*processing.py*): classe que encapsula el procés del classificador *Haar cascade* per trobar el cap de l'animal, explicat a la Secció 4.5. El codi font es pot consultar a l'Annex, Secció 2.2. Per a una descripció més acurada, consultar els propis *docstring* dins el codi en format *Sphinx*.

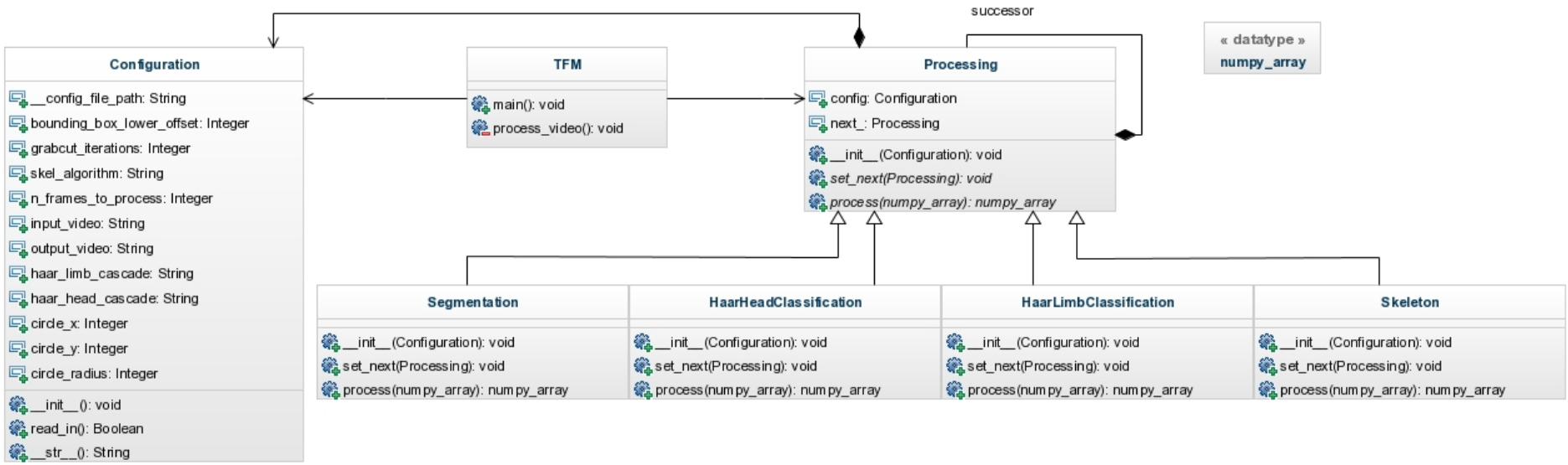


Figura 5.3: Diagrama de classes de disseny

- *HaarLimbClassification* (*processing.py*): classe que encapsula el procés del classificador *Haar cascade* per trobar les potes de l'animal, explicat a la Secció 4.5. El codi font es pot consultar a l'Annex, Secció 2.2. Per a una descripció més acurada, consultar els propis *docstring* dins el codi en format *Sphinx*.

Capítol 6

Resultats

Índex

En aquest capítol s'exposen els resultats obtinguts pel que fa a cada algorisme de forma individual així com la successiva combinació d'aquests que han permès arribar a la solució final. Pel que fa a les anotacions sobre el rendiment, cal tenir en compte que els temps de processat obtinguts han estat utilitzant el *hardware* següent:

- *CPU*: *Intel(R) Core(TM)2 Solo U3500 @1.40GHz*
- Memòria: *4GB*
- *SO*: *Debian 7 x86_64 GNU/Linux*

6.1 Segmentació

Com ja s'ha explicat prèviament, el procediment de segmentació utilitzat per arribar a la solució final ha estat un *Grab-cut* amb procediments extres per a automatitzar les parts que normalment solen ser assistides a l'hora d'utilitzar aquest algorisme. Aquest algorisme així com el procediment d'automatització es troben explicats a la Secció 4.4.2. En aquesta Secció el que es duu a terme és una mostra dels resultats que s'obtenen aplicant aquest procediment directament a cada fotograma, sense preprocessats ni postprocessats.

- Temps d'execució per fotograma: 2.3 segons
- Resultats en diferents fotogrames: Figura 6.1
- Alguns vídeos resultants d'aplicar només la segmentació:
    1. videos/segm_32.avi
    2. videos/segm_36.avi
    3. videos/segm_44.avi
    4. videos/segm_64.avi
    5. videos/segm_81.avi

Com es pot veure, en general la segmentació sol donar molt bons resultats a gairebé tots els fotogrames. En les situacions que es perden més extremitats és quan aquestes estan molt enganxades al cos, com quan l'animal es troba en repòs. Queda clar que, tot i els bons resultats, cal reforçar el mètode de segmentació perquè no es perdi cap extremitat com es veurà més endavant.

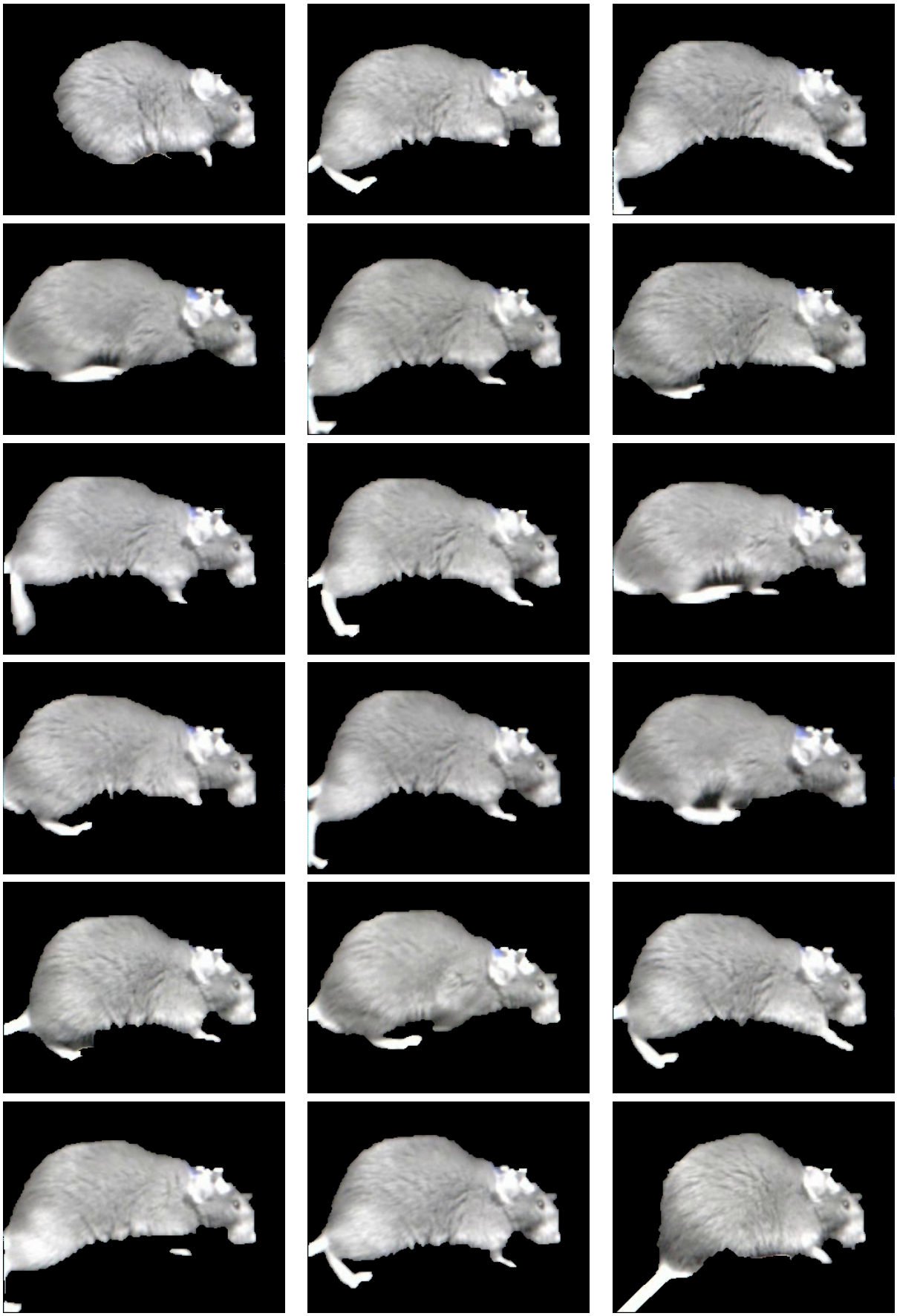


Figura 6.1: Resultats de segmentació en diferents fotogrames

6.2 Esquelet topològic

Un altre dels passos que s'han dut a terme per a localitzar les extremitats del ratolí ha estat intentar reconstruir l'esquelet topològic de l'animal. A més, com es veurà a continuació, és una de les operacions més costoses degut a que els algorismes estan desenvolupats íntegrament en *Python*. També hi té molt a veure que les proves s'hagin realitzat utilitzant el *hardware* mencionat a l'inici d'aquest capítol, bastant limitat.

- Temps d'execució per fotograma: 4.6 segons
- Resultats en diferents fotogrames: Figura 6.2
- Vídeos resultants d'aplicar únicament l'esquelet:
    1. videos/skel_64.avi
    2. videos/skel_81.avi

Queda clar que trobar l'esquelet topològic directament no serveix de gaire ja que existeix molt soroll que prové de la roda i les potes gairebé no es poden distingir en alguns casos. Així doncs, caldrà millorar aquesta cerca de l'esquelet a partir de la segmentació, imatges on únicament existeixi el cos del rosegador, tal i com es veurà a la Secció 6.3.

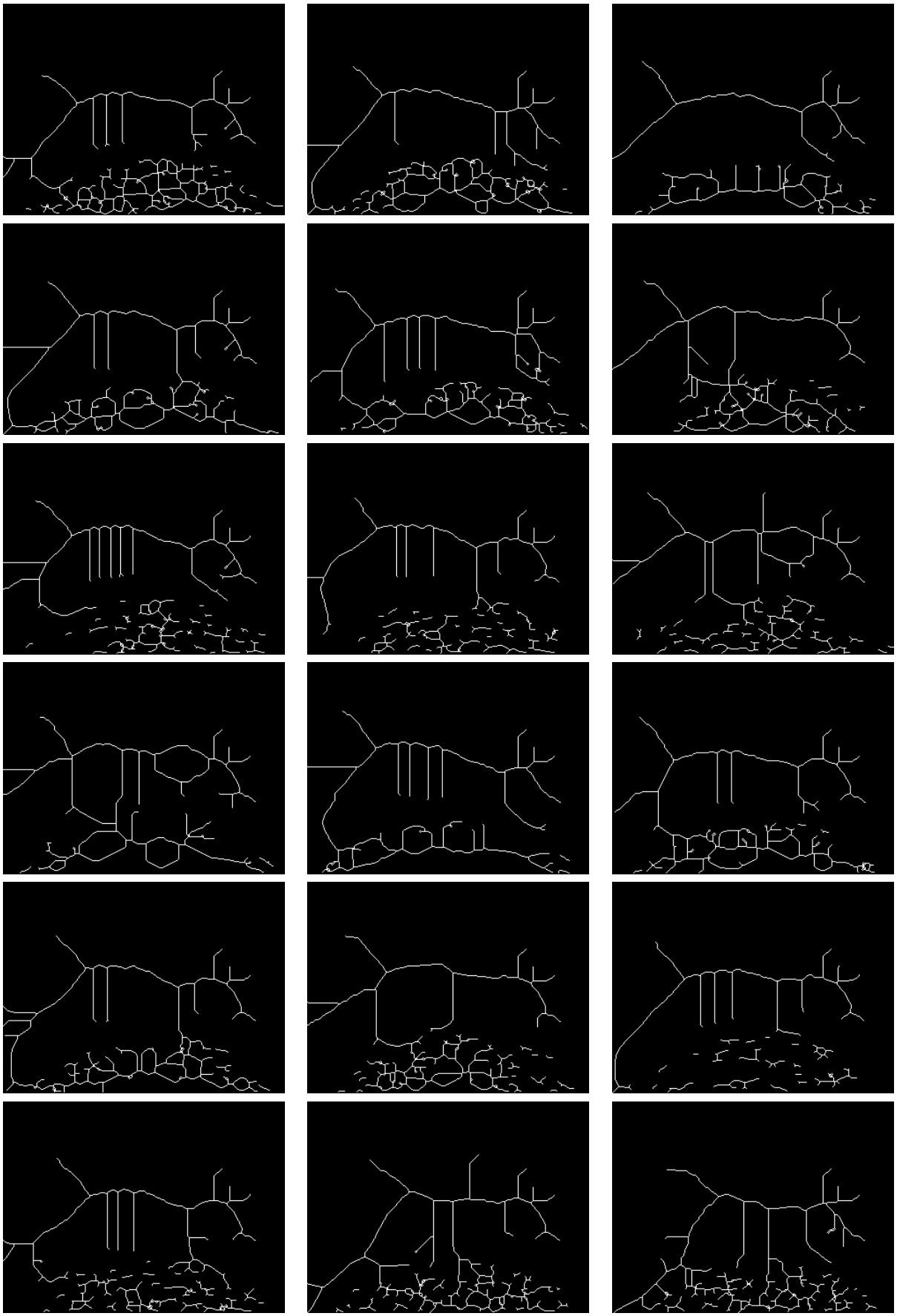


Figura 6.2: Resultats d'esquelet topològic en diferents fotogrames

6.3 Segmentació + esquelet

El resultat d'aplicar el mètode de construcció de l'esquelet topològic a partir de les imatges segmentades dóna molt millors resultats, tal i com es podrà apreciar a continuació. A més, la combinació dels dos mètodes no fa que els temps de processat vistos en les dues últimes seccions se sumin ja que l'imatge segmentada conté menys informació irrellevant i, en general, el mètode esdevé més eficient.

- Temps d'execució per fotograma: 5.3 segons
- Resultats en diferents fotogrames: Figura 6.3
- Vídeos resultants d'aplicar l'esquelet a la segmentació:
    1. videos/segm_skel_64.avi
    2. videos/segm_skel_81.avi

Aquests resultats juntament amb la segmentació inicial poden esser utilitzats en combinació amb els mètodes de classificació en cascada *Haar* per a marcar la posició de les potes, tal i com es veurà més endavant.

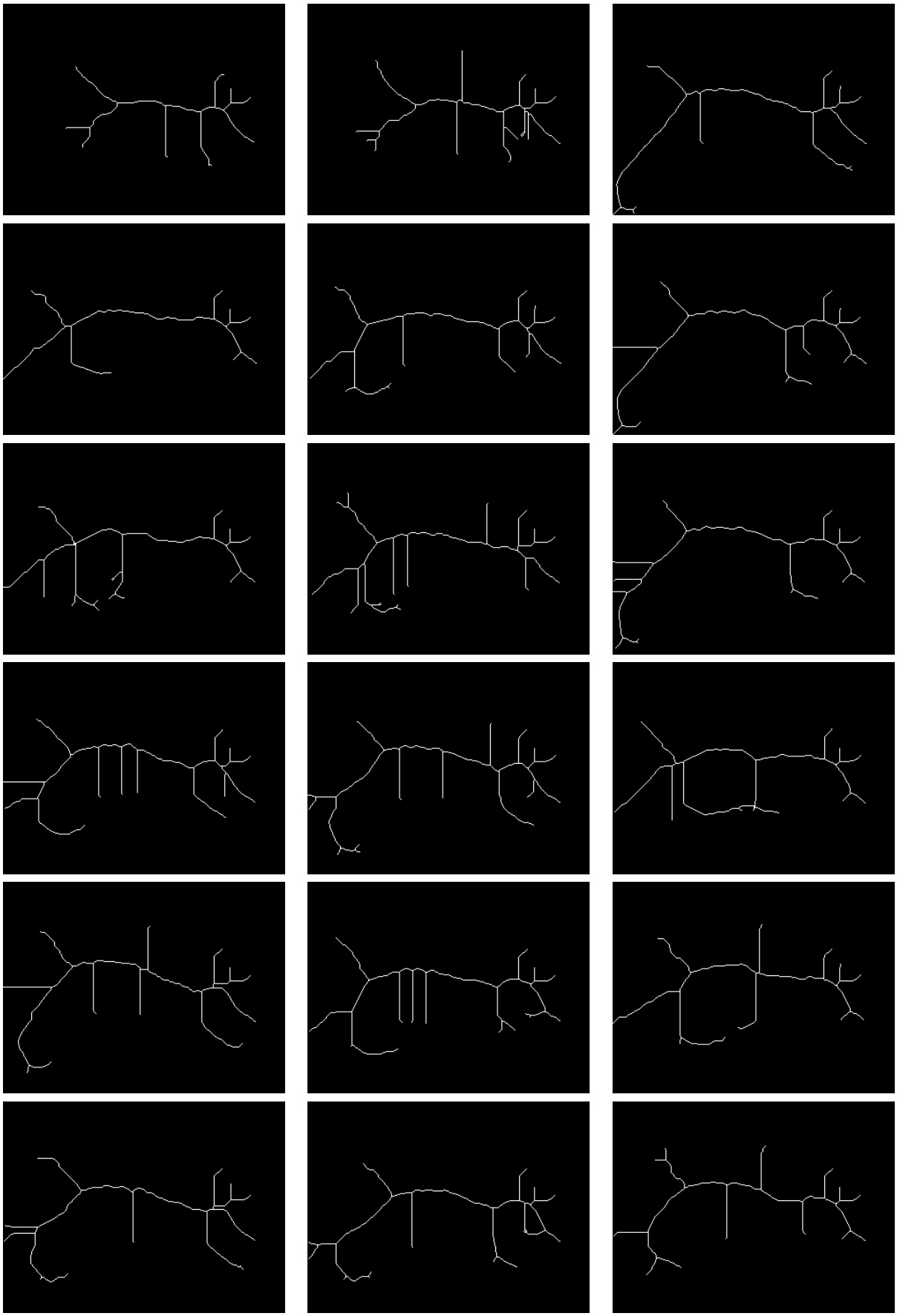


Figura 6.3: Resultats d'esquelet topològic sobre la segmentació en diferents fotogrames

6.4 Classificació *Haar* - extremitats

Els resultats de la cerca de les extremitats de l'animal utilitzant el classificador en cascada *Haar* ha acabat obtenint resultats sorprenentment bons, considerant que les potes són objectes molt deformables. Això és degut, sobretot, a l'entrenament del classificador dut a terme amb tants exemples positius i negatius tal i com s'explica a la Secció 4.5. Un punt fort d'aquesta classificació és el temps d'execució, que no sobrepassa els 0.03 segons utilitzant el *hardware* mencionat a l'inici d'aquest capítol (a costa de les hores que dura l'entrenament inicial del classificador).

- Temps d'execució per fotograma: 0.03 segons
- Resultats en diferents fotogrames: Figura 6.4
- Vídeos resultants d'aplicar el classificador *Haar* per trobar extremitats:
    1. Veure els resultats combinats de la secció 6.6.

Com ja s'ha mencionat, els resultats són molt bons i gairebé mai es perden potes, només quan aquestes es troben molt enganxades al cos o quan es troben molt aprop del límit del fotograma. La zona que queda marcada a les imatges d'exemple següents serviran per a marcar la posició de les potes, juntament amb la informació obtinguda de la segmentació i de l'esqueletització tal i com es veurà més endavant.

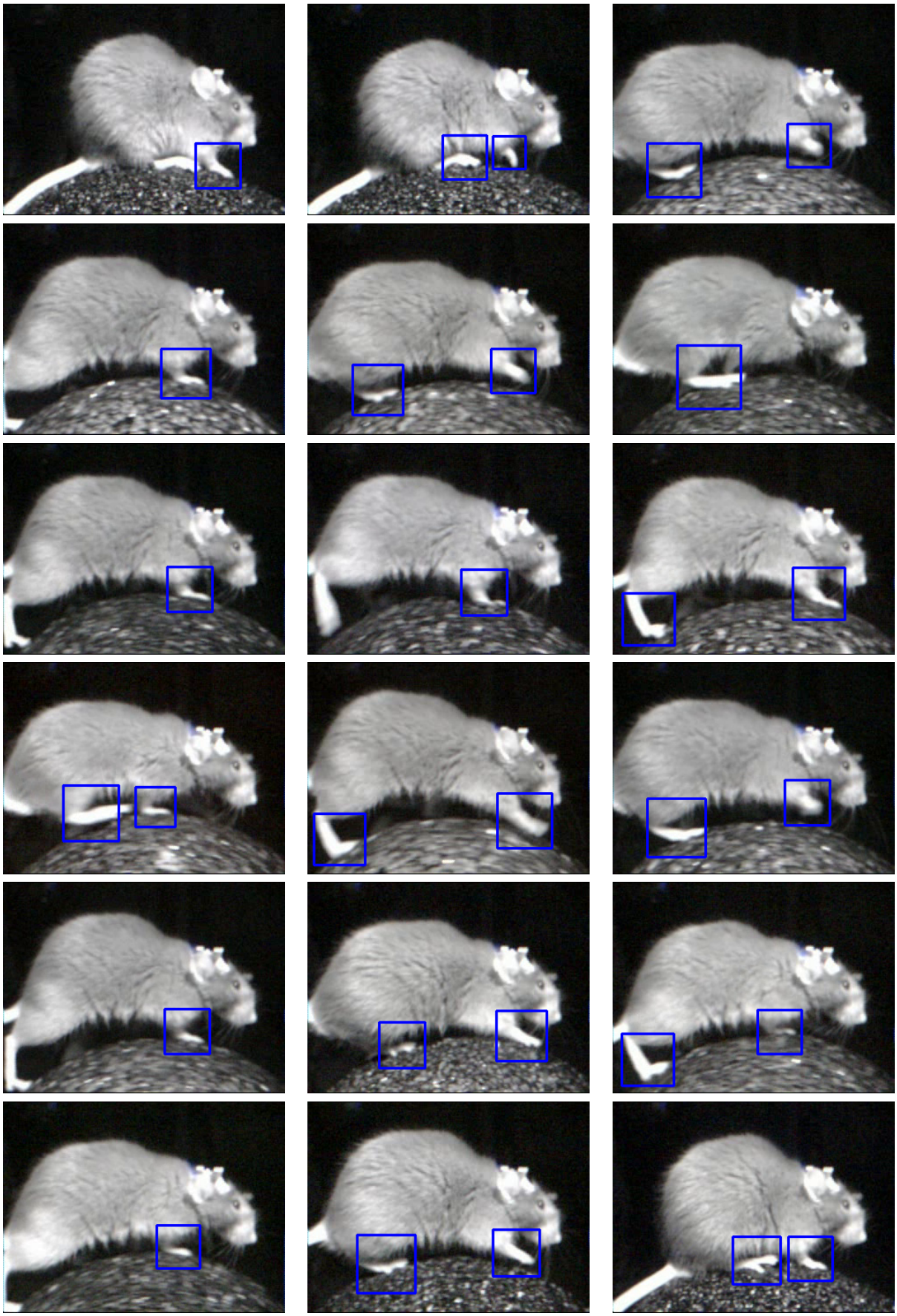


Figura 6.4: Resultats de la classificació *Haar* d'extremitats en diferents fotogrames

6.5 Classificació *Haar* - cap

La classificació del cap de l'animal dóna un encert del 100% en tots els vídeos de prova que s'han utilitzat per a la realització d'aquest treball. Això és degut al bon entrenament que s'ha fet del classificador en cascada *Haar* i també que el cap és un objecte molt poc deformable i estable que gairebé és idèntic a cada fotograma. Els temps de cerca són gairebé menyspreables, al voltant de la centèsima de segon.

- Temps d'execució per fotograma: 0.015 segons
- Resultats en diferents fotogrames: Figura 6.5
- Vídeos resultants d'aplicar el classificador *Haar* per trobar el cap:
    1. Veure els resultats combinats de la secció 6.6.

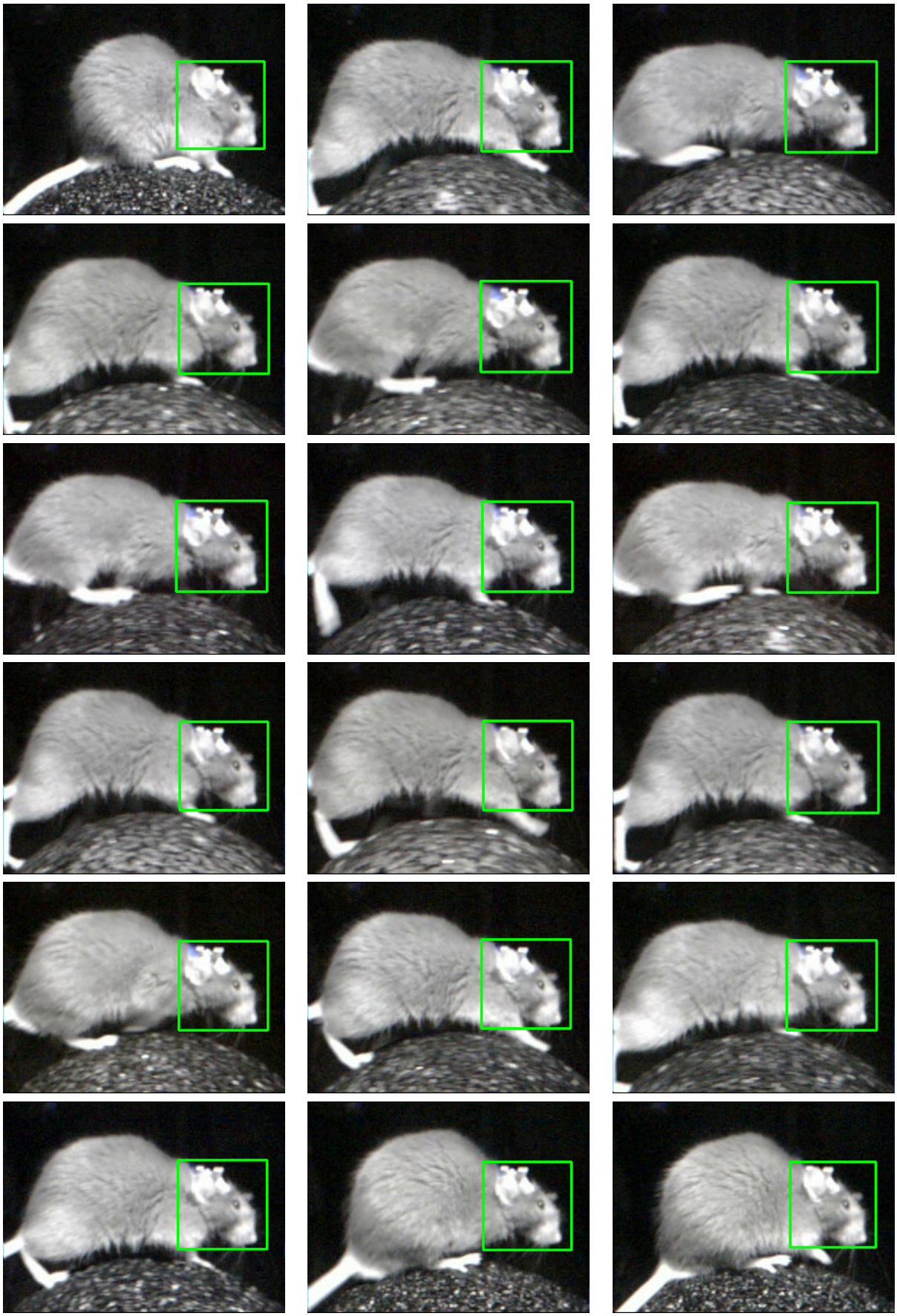


Figura 6.5: Resultats de la classificació *Haar* del cap en diferents fotogrames

6.6 Classificació *Haar* - cap i extremitats

La combinació dels dos classificadors dóna, com és evident, el mateix resultat obtingut individualment però combinat. Els temps d'execució es sumen, en aquest cas un no facilita la feina a l'altre ni en millora el resultat com si passa amb la combinació de la segmentació i l'esqueletització.

- Temps d'execució per fotograma: 0.038 segons
- Resultats en diferents fotogrames: Figura 6.6
- Vídeos resultants d'aplicar el classificador *Haar* per trobar el cap i les extremitats:
    1. videos/haar_32.avi
    2. videos/haar_36.avi
    3. videos/haar_44.avi
    4. videos/haar_64.avi
    5. videos/haar_81.avi

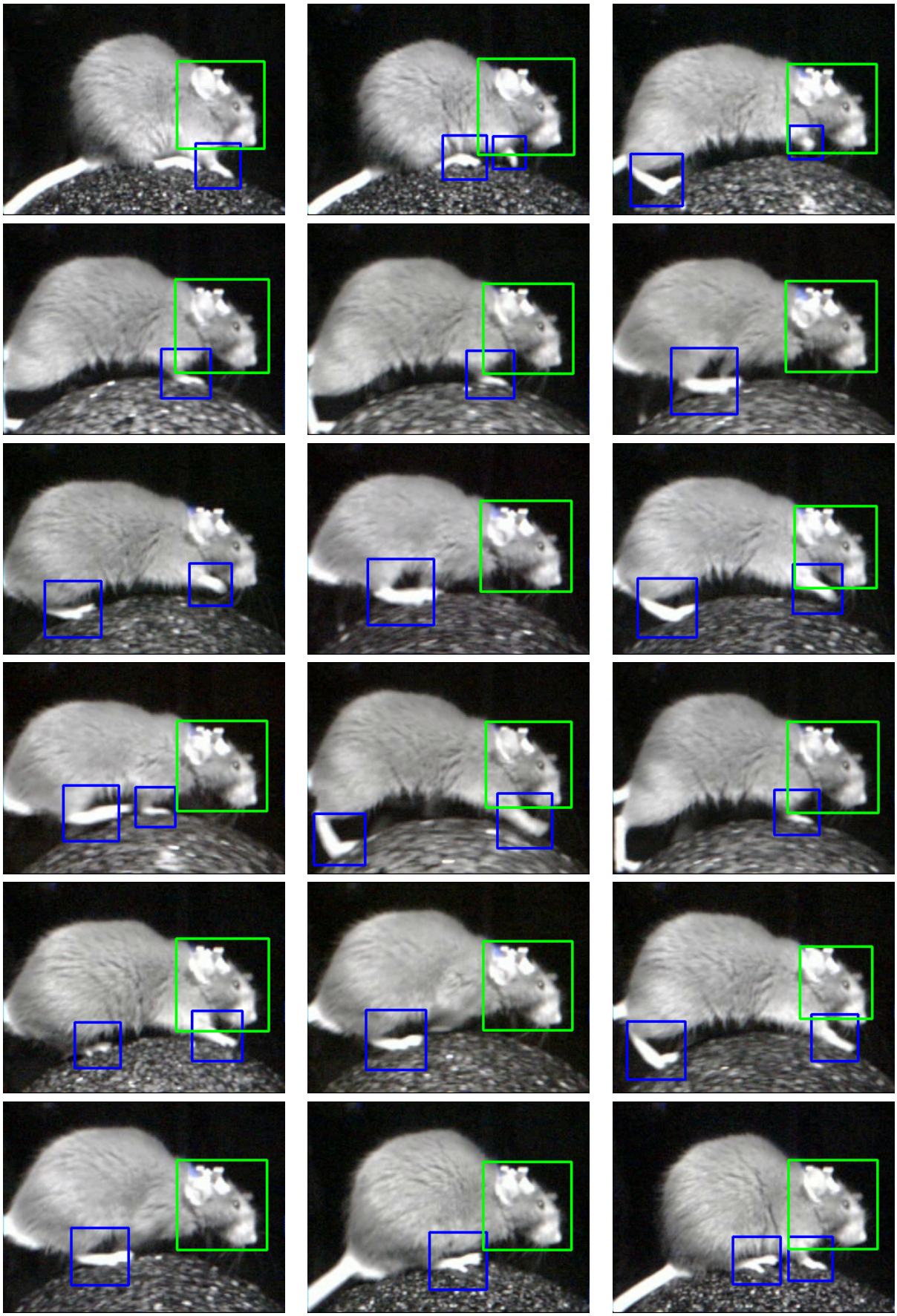


Figura 6.6: Resultats de la classificació *Haar* del cap i les extremitats en diferents fotogrames

6.7 Solució final

La solució final consisteix en aprofitar tota la informació produïda pels diferents algorismes de visió artificial anteriors i combinar-la mitjançant el procediment que s'explicarà en aquesta Secció, resultant en el codi final que es pot consultar a l'Annex, Secció 2, el manual d'usuari del qual també es pot trobar a l'Annex, Secció 2.4. Primer de tot, per obtenir les dades requerides s'han definit dues cadenes d'execució diferents partint de la mateixa imatge d'origen, una que durà a terme el procés de segmentació i l'altra que durà a terme el procés de classificació. Aquest flux de dades queda visualment representat en el diagrama de la Figura 6.7 i es recomana anar-lo seguint a mesura que es duu a terme l'explicació de cada branca per entendre-ho millor. Primer de tot, vegem la descripció de la cadena de processos pas a pas:

- Branca de segmentació (camí superior de la Figura 6.7): aquesta cadena d'algorismes s'encarrega d'obtenir la segmentació del ratolí i seguidament de trobar l'esquelet topològic d'aquest, essent aquest ordre important:

    1. Segmentació *Grab-cut* automatitzada: el primer algorisme d'aquesta cadena és el que es troba explicat a la Secció 4.4.2, l'automatitzat, i que proporciona resultats semblants als mostrats a la Secció 6.1. Intenta obtenir la segmentació de l'animal per a ser aprofitada pels pas següent i per la combinació final.
    2. Esquelet topològic del resultat de la segmentació: el següent pas de la cadena superior d'algorismes consisteix en construir l'esquelet del cos segmentat utilitzant un dels tres algorismes descrits a la Secció 4.3. Per a la solució final s'ha utilitzat l'implementat utilitzant *SciPy*, degut a que el temps d'execució és més ràpid que el dels altres dos. Els resultats que obté es mostren a la Secció 6.3 i són els que s'aprofiten per la solució final.

- Branca de classificació (camí inferior de la Figura 6.7): aquesta cadena d'algorismes s'encarrega de localitzar la posició de les extremitats i del cap, proporcionant un *ROI* (regió d'interès) on es combinarà la informació trobada per la primera branca per marcar una posició més exacta, si és possible:

    1. Classificació *Haar* d'extremitats: el primer algorisme d'aquesta cadena s'encarrega de localitzar la posició de les extremitats de l'animal i proporcionar un *ROI* que les contingui. Per a fer això s'ha entrenat un classificador *Haar* en cascada tal i com s'explica a la Secció 4.5 utilitzant molts exemples de potes. El procediment de classificació utilitzant el classificador entrenat proporciona resultats semblants

als que es poden veure a la Secció 6.4 i, com ja s'ha mencionat, els *ROI* resultants seran utilitzats al pas final per afinar la detecció utilitzant la informació de l'imatge segmentada i l'esquelet d'aquesta.

2. Classificació *Haar* del cap: el següent algorisme consisteix en aplicar un altre classificador *Haar* encarregat de localitzar el cap de l'animal però al final ha acabat essent de poca importància ja que no s'ha utilitzat com a informació per afinar la localització de les extremitats (no ha estat necessari). De totes maneres, els resultats obtinguts són els que es poden veure a la Secció 6.5 i es disposa d'aquesta informació per si es vol utilitzar en un futur al pas final.

Un cop executades les cadenes d'algorismes, la informació produïda es combina de la següent manera per obtenir el resultat final per cada fotograma:

1. Per cada *ROI* d'extremitat trobada, es crea una màscara per marcar millor la pota i localitzar l'extrem inferior de cada una d'elles:

   (a) S'agafa la informació d'esquelet que queda dins el *ROI*, s'elimina qualsevol *blob* que no sigui el més gran i es passen dues iteracions *dilate* amb una finestra de *3x3* per fer la línia més gruixuda i que es vegi millor.

   (b) Dins el mateix *ROI*, es combina la informació de l'esquelet que s'acaba de dilatar juntament amb la de la segmentació i la imatge original utilitzant una *and* lògica. Això fa que només quedin els píxels que són presents en les tres capes i resulta en una regió on només hi ha marcat l'esquelet de la pota. Amb això, en les imatges on hi ha hagut una bona segmentació, duu a una localització molt bona i la pota queda molt ben marcada.

2. Un cop es disposa d'aquesta màscara, es localitza el punt de més a sota a la dreta i es marca com la punta de l'extremitat. Aquesta informació es guarda a posteriori dins un arxiu *CSV*, juntament amb les coordenades del *ROI*, per a poder saber a posteriori la localització de cada pota en cada fotograma per cada vídeo.

3. S'aplica la màscara sobre l'imatge original, marcant la pota trobada en vermell una mica transparent per continuar veient l'extremitat real. L'imatge original s'enfosqueix una mica perquè el vermell transparent ressalti més.

4. Es dibuixa el *ROI* on s'ha localitzat l'extremitat i el punt on es considera que s'ha trobat el punt inferior de l'extremitat.

Seguint aquests passos, s'obté el resultat final de la localització de les extremitats tal i com es pot veure a la Figura 6.8. Originalment també es construïa una màscara per marcar la localització del cap en blau transparent tal i com es pot veure a la Figura 6.9 però al final es va acabar descartant degut a que no aportava informació valuosa.

Així doncs, vegem alguns exemples de la solució final en imatges i també en tots els vídeos processats:

- Temps d'execució per fotograma (en combinar resultats): 0.007 segons
- Resultats en diferents fotogrames: Figures 6.10, 6.11 i 6.12
- Vídeos resultants d'aplicar tots els algorismes i fer la combinació final:
    1. videos/final_Vid1.avi
    2. videos/final_Vid10.avi
    3. videos/final_Vid14.avi
    4. videos/final_Vid32.avi
    5. videos/final_Vid36.avi
    6. videos/final_Vid44.avi
    7. videos/final_Vid64.avi
    8. videos/final_Vid81.avi
    9. videos/final_Vid36_slow_framerate.avi

El vídeo *videos/final_Vid36_slow_framerate.avi* s'ha generat per poder visualitzar més lentament el resultat de la detecció i poder-ne apreciar millor els detalls.

Tal i com es pot veure, el resultat obtingut en els tres primers vídeos és més pobre que amb la resta (la classificació funciona igual de bé però la segmentació no tant i això fa que no es marquin tant les potes). Això és degut a que les condicions d'il·luminació són diferents i a que el ratolí es troba en una posició diferent que la resta de vídeos, els que es van utilitzar originalment per entrenar i segmentar; caldria adaptar millor els paràmetres de configuració per obtenir una segmentació més eficient. A la Secció 6.7.1 es parla més a fons dels diferents tipus de detecció que poden existir, depenent dels resultats obtinguts en cada algorisme de la cadena.

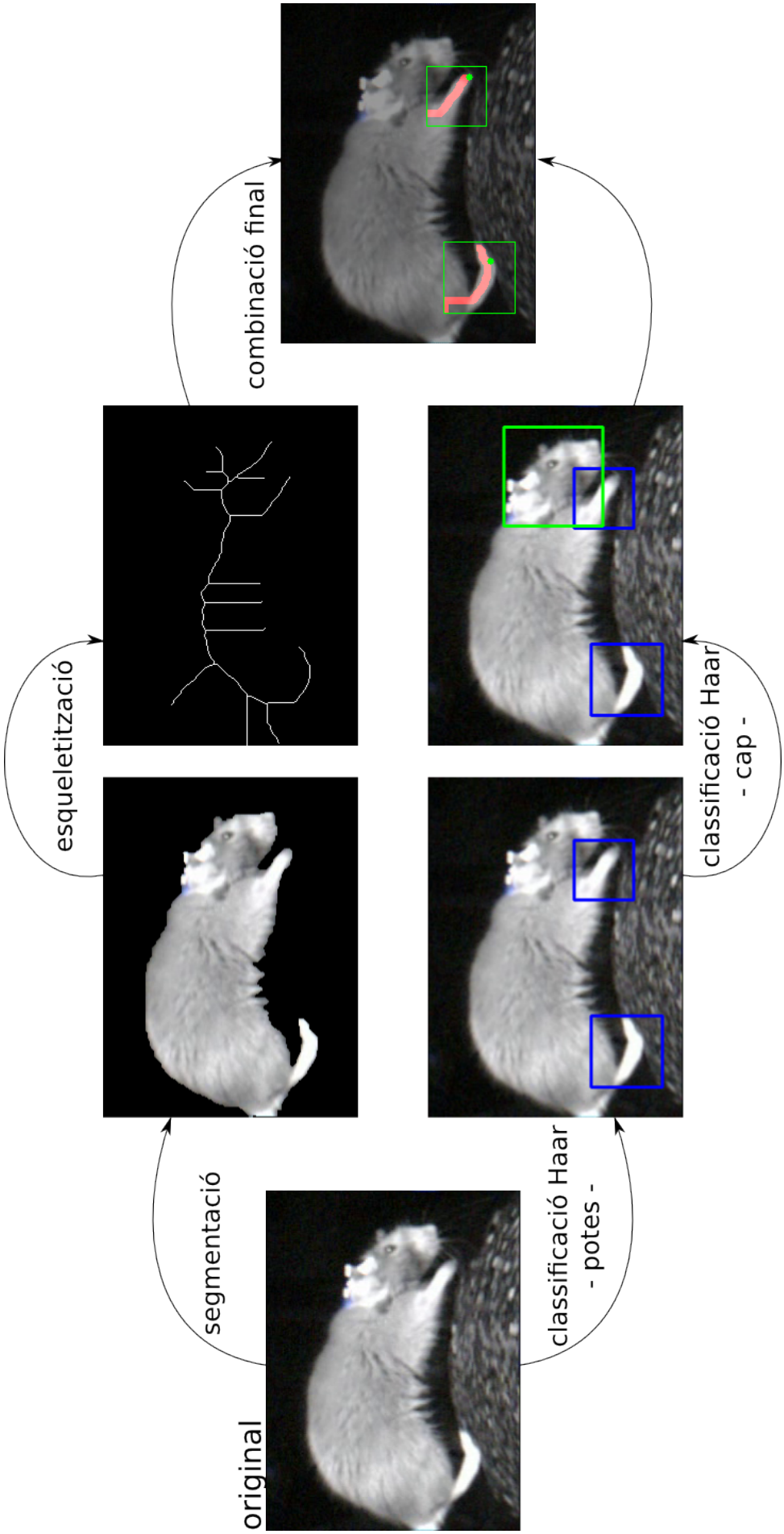


Figura 6.7: Procés de combinació dels resultats en cadena per arribar a la solució final

A continuació es pot veure el resultat obtingut en un sol fotograma i s'explica què significa el que es marca en cada un d'ells al final del processat:

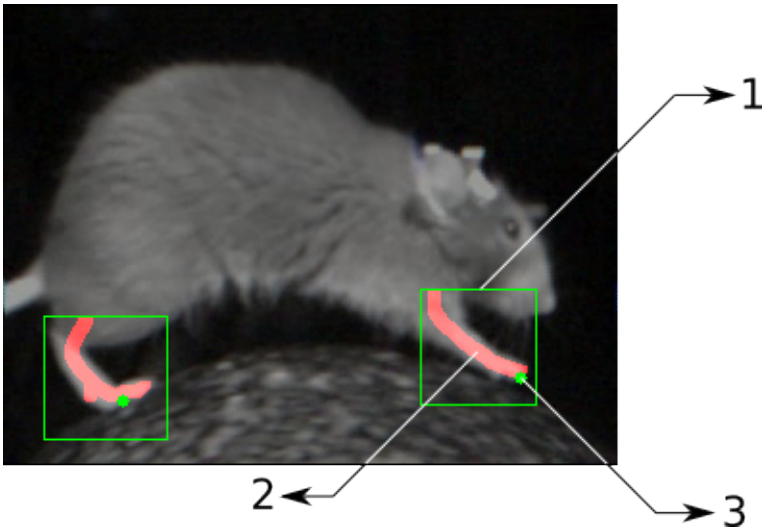


Figura 6.8: Exemple processat final

1. Es tracta del *ROI* trobat pel classificador *Haar* de potes, la regió on es duu a terme el processat final.

2. Es tracta de la combinació de l'esquelet dilatat amb la segmentació, marcat de color vermell transparent per mostrar la detecció final de l'extremitat.

3. El punt verd marca la posició on es considera que es troba la punta de l'extremitat en aquest *ROI*.

Com ja s'ha comentat anteriorment, originalment es marcava també la posició del cap en blau transparent però es va acabar descartant per la solució final. Un exemple de com quedava originalment es pot veure a continuació:

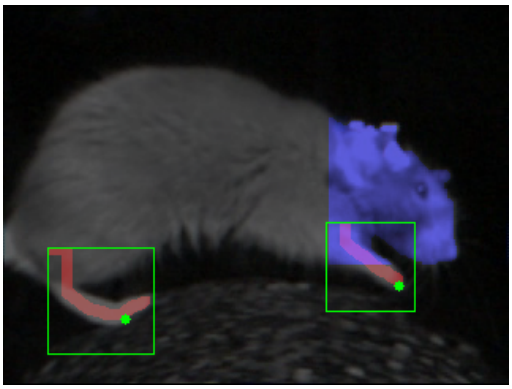


Figura 6.9: Exemple processat final marcant la posició del cap - descartat

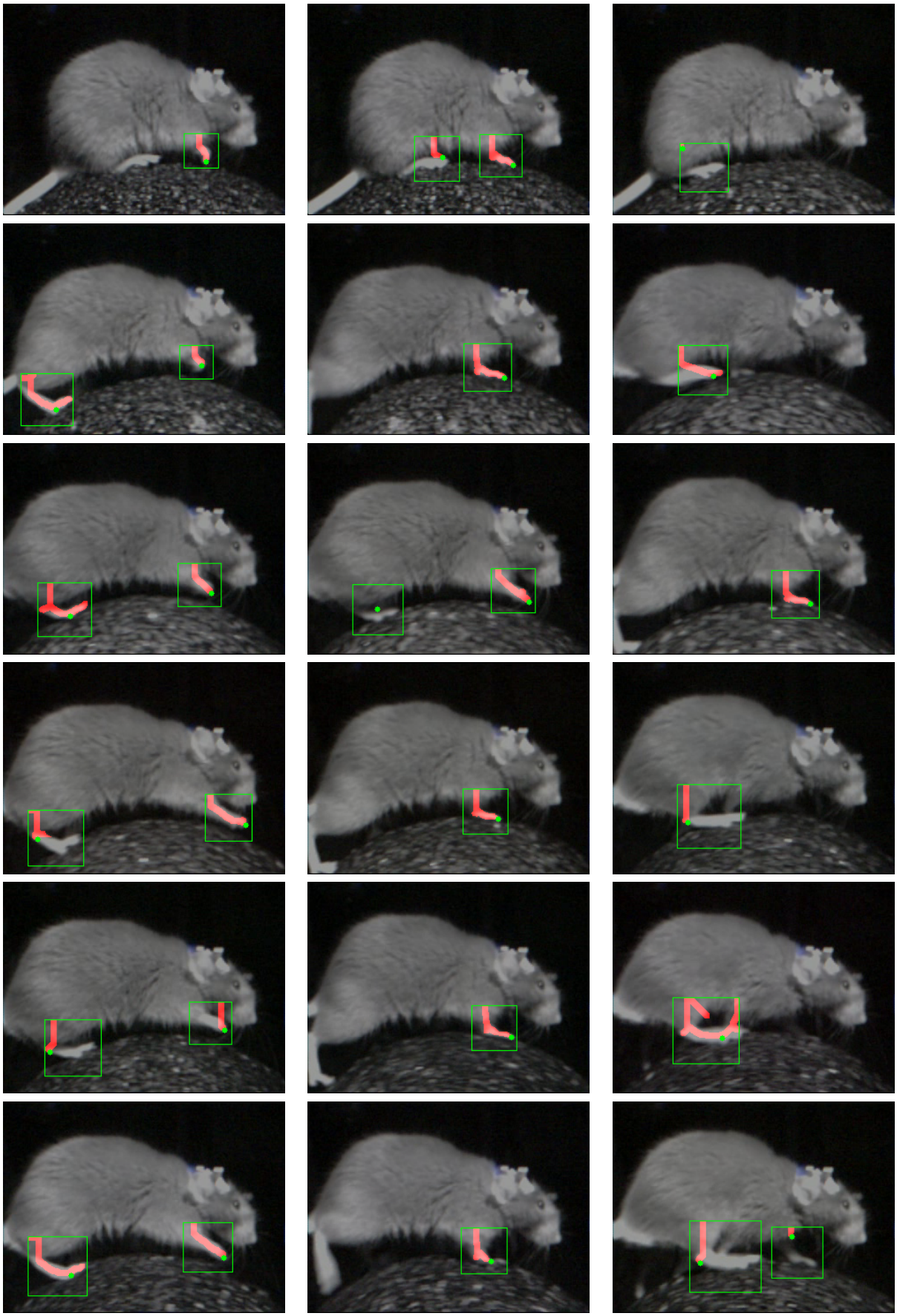


Figura 6.10: Resultats de la solució final en diferents fotogrames - I

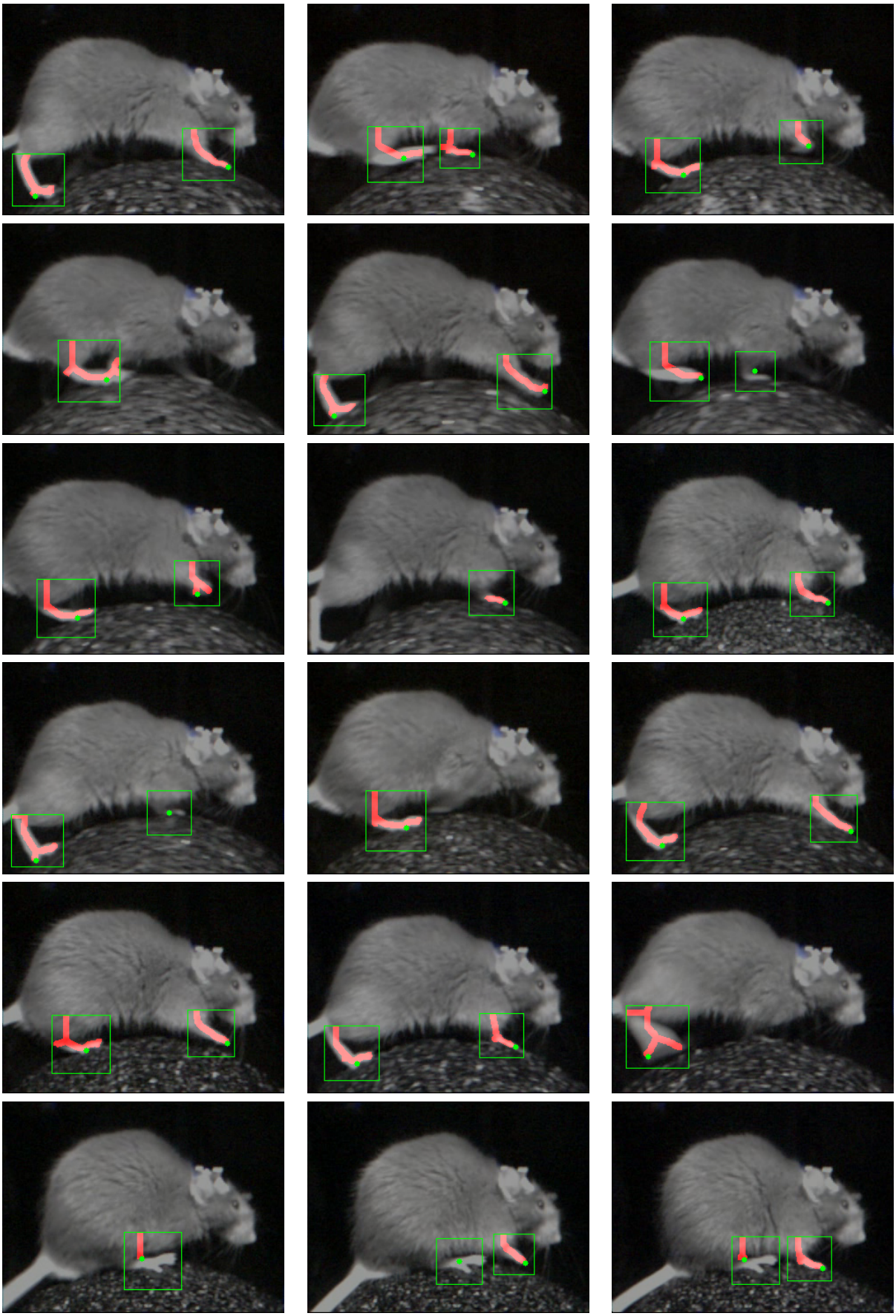


Figura 6.11: Resultats de la solució final en diferents fotogrames - II

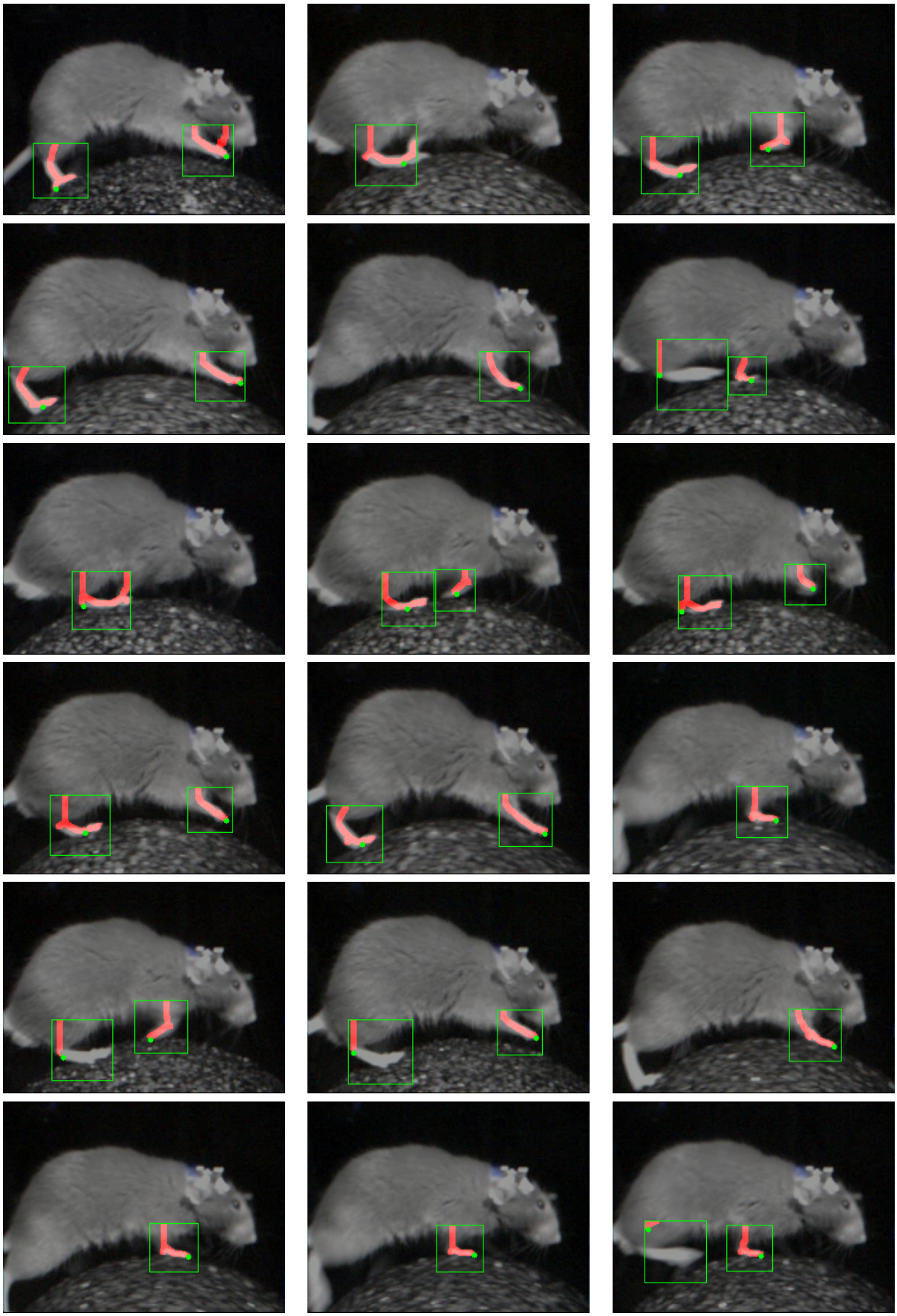


Figura 6.12: Resultats de la solució final en diferents fotogrames - III

Com es pot veure, la detecció és bastant bona en molts dels fotogrames. A la Secció 6.7.1 es fa una anàlisi dels possibles tipus de detecció que es poden observar als resultats finals.

A més, com ja s'ha comentat anteriorment, les coordenades de la localització final de cada extremitat per cada fotograma queden emmagatzemades en un arxiu *CSV* per a ser analitzades a posteriori. Aquests arxius serveixen per a saber la posició de cada pota localitzada per cada fotograma de cada vídeo processat i segueixen la nomenclatura següent:

- *results_nomvídeo_YYYYmmddHHMMSS.csv*

A dins s'hi van emmagatzemant les següents propietats, separades per columnes:

- *frame*: nombre de fotograma.
- *limb1_x*: posició x de la primera extremitat detectada.
- *limb1_y*: posició y de la primera extremitat detectada.
- *limb1_roi_x*: posició x del *ROI* on s'ha detectat la primera extremitat.
- *limb1_roi_y*: posició y del *ROI* on s'ha detectat la primera extremitat.
- *limb1_roi_w*: amplada del *ROI* on s'ha detectat la primera extremitat.
- *limb1_roi_h*: llargada del *ROI* on s'ha detectat la primera extremitat.
- *limb2_x*: posició x de la segona extremitat detectada.
- *limb2_y*: posició y de la segona extremitat detectada.
- *limb2_roi_x*: posició x del *ROI* on s'ha detectat la segona extremitat.
- *limb2_roi_y*: posició y del *ROI* on s'ha detectat la segona extremitat.
- *limb2_roi_w*: amplada del *ROI* on s'ha detectat la segona extremitat.
- *limb2_roi_h*: llargada del *ROI* on s'ha detectat la segona extremitat.

Aquesta informació permet dibuixar gràfics de posició on es pot intuir la zona de moviment de cada extremitat com els que es mostren a continuació pel vídeo *PS3_Vid81.avi*:

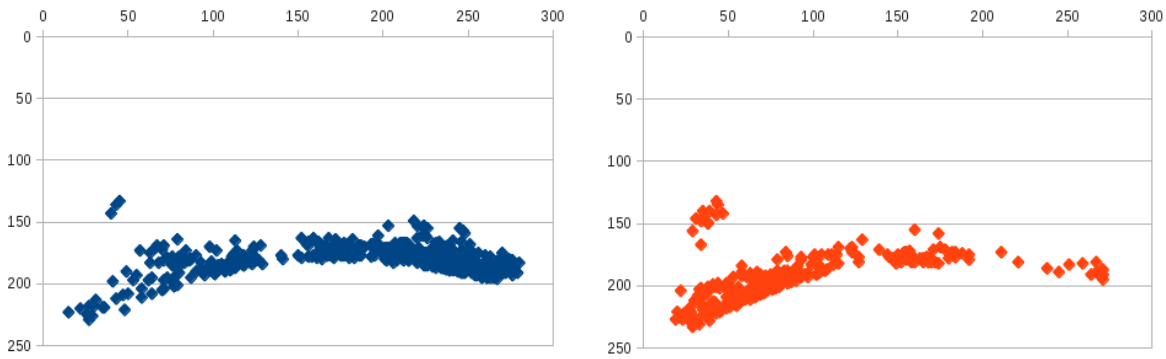


Figura 6.13: Gràfics de posició - *PS3_Vid81.avi*

A l'esquerra es dibuixa la informació agafada de les columnes *limb1_x* i *limb1_y* i a la dreta la informació agafada de la columna *limb2_x* i *limb2_y*.

En cap moment s'ha fet distinció sobre si l'extremitat detectada és la del darrere o la del davant, simplement es detecten les extremitats i, a l'esquerra, es pinta la informació de la primera extremitat detectada i, a la dreta, la de la segona. Es pot apreciar com, a l'esquerra, hi predomina la detecció de la pota del davant (el núvol de punts és molt més intens a la part del davant, entre on hi hauria la roda i el cap) i, a la dreta, hi predomina la detecció de la pota del darrere (el núvol de punts és molt més intens entre la roda i on seria la cua). Hi ha casos on només es detecta una sola extremitat o on l'ordre de la detecció no és el mateix i això fa que als dos gràfics hi hagi presència de les dues extremitats (a més de valors clarament *outliers* com els de la pota del darrere tan amunt). Una característica a destacar és que es pot veure clarament com es ressegueix el contorn de la bola, on les potes hi estan en contacte durant bona part de l'estona.

El fet que no es diferenciïn les potes del davant de les del darrere fa que tenir dos gràfics separats per representar la posició de les potes al llarg del vídeo no sigui gaire valuós i sigui preferible ajuntar la informació d'ambdós per obtenir el gràfic final de moviment:

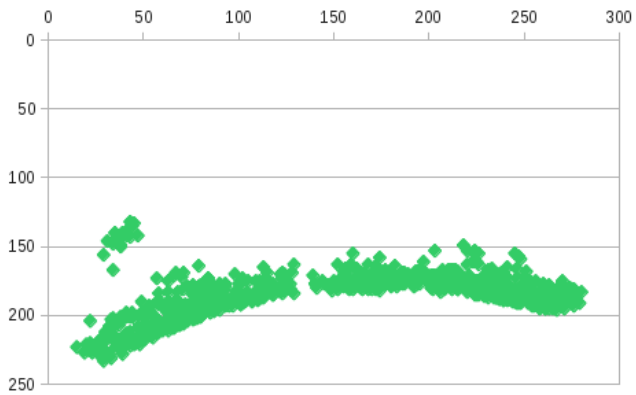


Figura 6.14: Gràfics de posició combinats - *PS3_Vid81.avi*

Aquí es pot veure clarament la regió per on es mou cada extremitat, marcada per dos clústers, i es podria arribar a diferenciar fàcilment fins i tot amb algun algorisme d'Intel·ligència Artificial simple, com el *K-means*. Vegem els resultats obtinguts amb alguns altres vídeos:

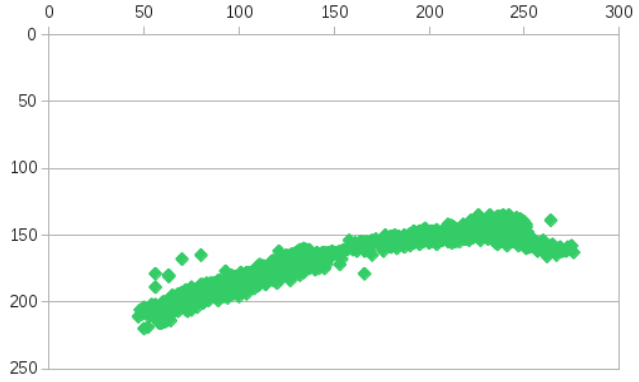


Figura 6.15: Gràfics de posició combinats - *PS3_Vid1.avi*

El vídeo *PS3_Vid1.avi* presenta el ratolí en una posició diferent, la roda es troba més cap a la dreta i, degut a això, la pota posterior es troba més a sota que la frontal. Degut a les condicions d'il·luminació diferents, la segmentació no és gaire bona en aquest vídeo i les deteccions surten més del punt mig de la classificació, sense ser afinades finalment.

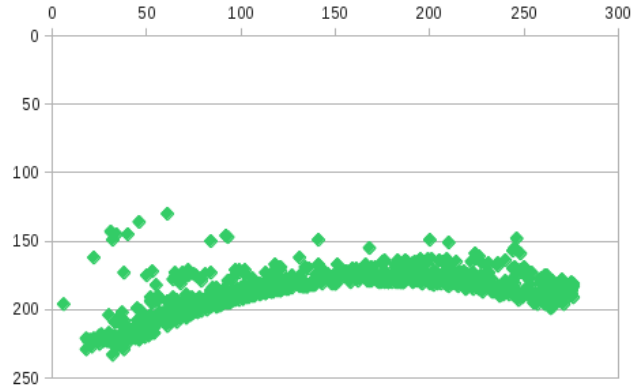


Figura 6.16: Gràfics de posició combinats - *PS3_Vid32.avi*

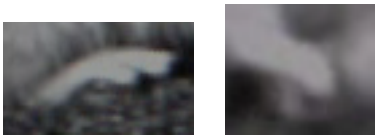
El vídeo *PS3_Vid32.avi* presenta unes característiques semblants al primer de tots i, en conseqüència, la gràfica de posicionament s'hi assembla molt.

6.7.1 Possibles tipus de detecció

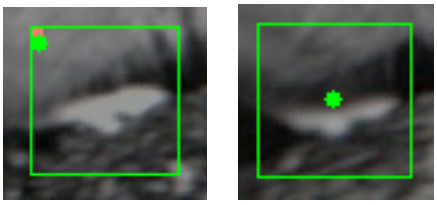
Com ja s'ha vist a les captures de diversos resultats agafats de fotogrames aleatoris i com ja s'ha anat comentant, existeixen vàries modalitats de detecció que val la pena conèixer per a

entendre el motiu pel que han donat el resultat observat:

- Sense resultat de classificació: si per cert fotograma no s'ha trobat la posició d'una extremitat a partir de la classificació *Haar*, no existirà detecció d'aquesta. Els exemples d'aquest cas són evidents, simplement no existeix detecció:

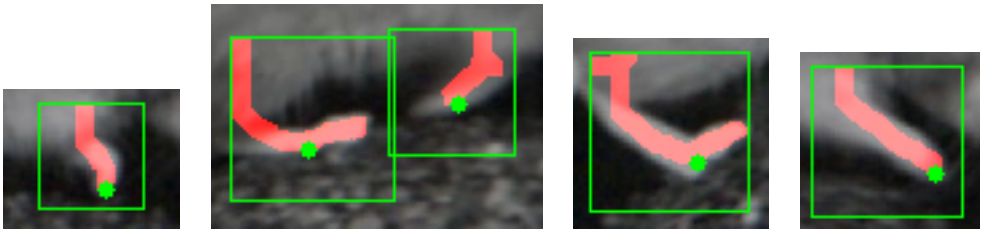


- Existeix resultat de classificació però la segmentació no ha estat bona: en aquests casos, si no es pot trobar res dins el *ROI* (ni segmentació ni, evidentment, esquelet), es marca com a detecció el punt mig de la zona trobada. Això fa que, sovint, tot i no haver-hi segmentació la localització sigui molt bona:



  En altres casos, com el de la dreta, pot passar que la segmentació i l'esqueletització no hagi fallat del tot i es marquin zones que realment no són la pota.

- Encerts: en cas que la segmentació hagi estat bona, l'esqueletització també ho serà i, si també s'ha localitzat la posició de la pota, s'obtindrà un resultat positiu:



6.8 Validació dels resultats

Per a ser capaços de validar els resultats obtinguts a partir de la solució final, ha calgut establir certs protocols per a determinar quin grau d'èxit s'ha assolit. Evidentment, no es disposa d'un mètode automàtic per a determinar on es troben les potes del ratolí dins l'entorn de prova per poder comparar resultats ja que, si fos així, la feina feta en aquest treball ja no seria necessària. Per tant, l'únic mètode per poder validar els resultats ha de passar, per força, a través de l'actuació manual i visual de l'usuari.

Així doncs, s'ha desenvolupat una eina auxiliar que permet demanar interactivament a l'usuari si accepta com a bones o no les posicions detectades a cada fotograma per cada vídeo processat i extreure d'aquesta informació les estadístiques pertinents. El codi d'aquesta utilitat es pot trobar a la Secció 4 de l'Annex o al codi font entregat juntament amb aquest informe. A continuació se'n mostra el funcionament:

L'eina es crida des de línia de paràmetres de següent manera:

```
python haar_sampler.py
```

Es poden utilitzar certs paràmetres per a especificar el vídeo que cal analitzar i el *path* on cal guardar l'arxiu amb els resultats finals:

```
python haar_sampler.py -v <video_file> -p <results_path>
```

- *-v*: permet especificar el vídeo que cal analitzar.
- *-p*: permet especificar el *path* on cal guardar l'arxiu amb els resultats finals.

Un cop en marxa, l'eina mostra la interfície que es pot veure a la Figura 6.17 i l'usuari ja pot començar a validar els resultats. Per a tancar l'aplicació en qualsevol moment, cal prémer la tecla d'escapament i per a avançar al següent fotograma cal prémer la tecla *'n'*.

Un cop analitzat tot el vídeo processat, l'aplicació genera un arxiu *.csv* amb les respostes de l'usuari que podrà esser utilitzat per al seu posterior anàlisi o per a generar estadístiques i gràfiques. La informació que es guarda dins l'arxiu comença amb una línia comentada on s'informa del nom del vídeo analitzat i després s'insereix la informació de l'anàlisi seguint l'ordre següent:

Frame	*Rear limb*	*Front limb*

On es pot veure el nombre de fotograma, si s'accepta la posició de l'extremitat posterior i si s'accepta la posició de l'extremitat del davant.

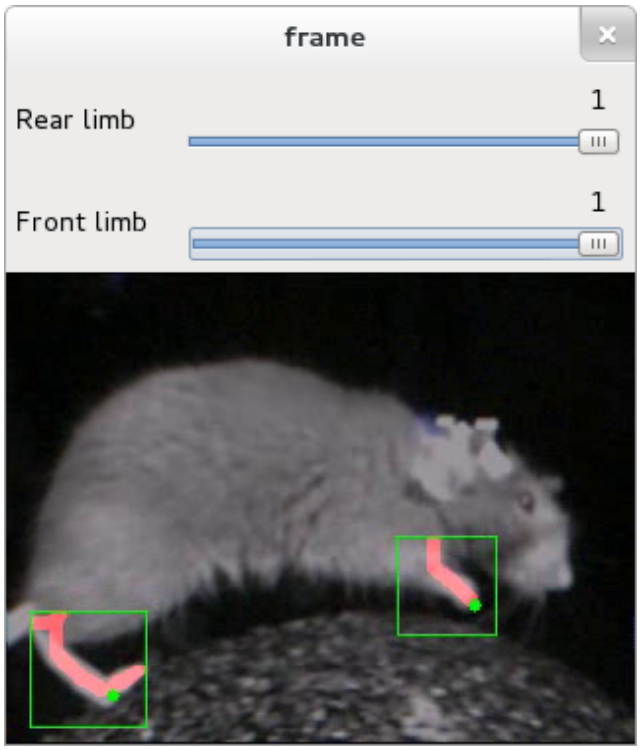


Figura 6.17: Interfície de l'aplicació de validació

A continuació es mostren els resultats obtinguts pels diferents vídeos analitzats:

- *final_Vid1.avi*:
  - Percentatge d'encerts (extremitat posterior): 73%
  - Percentatge d'encerts (extremitat frontal): 80%
  - Fotogrames analitzats: 767

  Aquest vídeo té un percentatge molt baix de segmentacions ben fetes però el percentatge de classificació és molt elevat així que arriba fins al 73% d'encerts pel que fa a deteccions de l'extremitat posterior i al 80% pel que fa a les potes del davant. Cal tenir en compte, però, que a aquests primers vídeos es poden veure en molts casos les quatre extremitats alhora i s'ha considerat com a bona la detecció d'ambdues o una de cada. El percentatge d'encerts pel que fa a les potes que es troben en primer pla és més baix però localitzar les del fons també s'ha considerat correcte.

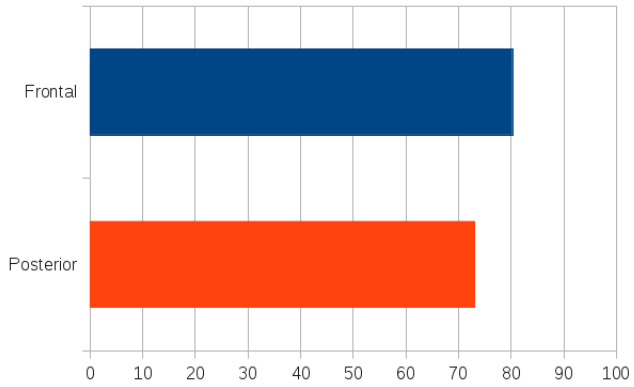


Figura 6.18: Percentatges d'encerts - *Vid1*

- *final_Vid10.avi*:
  - Percentatge d'encerts (extremitat posterior): 81%
  - Percentatge d'encerts (extremitat frontal): 78%
  - Fotogrames analitzats: 767

  Aquest vídeo és de les mateixes característiques que l'anterior i, per tant, es poden apreciar les quatre potes en molts instants del vídeo. S'ha considerat com encert en cas de classificar una o ambdues potes frontals i posteriors. En aquest cas l'encert de potes frontals és superior al de potes posteriors, al contrari que el vídeo anterior, però per poc.

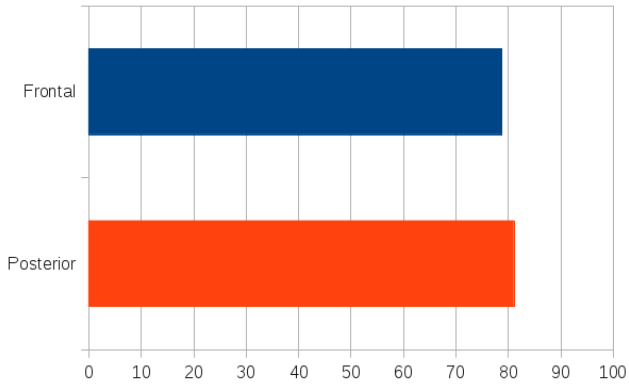


Figura 6.19: Percentatges d'encerts - *Vid10*

- *final_Vid14.avi*:
  - Percentatge d'encerts (extremitat posterior): 71%

- Percentatge d'encerts (extremitat frontal): 78%
- Fotogrames analitzats: 767

Aquest vídeo també és de les mateixes característiques que els dos anteriors i, per tant, els resultats són semblants.

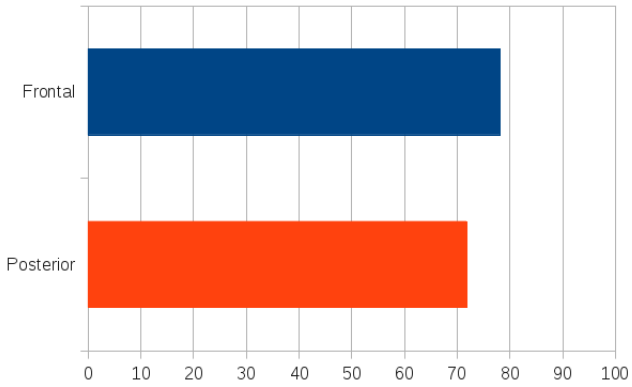


Figura 6.20: Percentatges d'encerts - *Vid14*

- *final_Vid32.avi*:
  - Percentatge d'encerts (extremitat posterior): 85%
  - Percentatge d'encerts (extremitat frontal): 82%
  - Fotogrames analitzats: 479

El percentatge d'encerts en aquest vídeo és realment molt elevat, més del que indiquen les estadístiques. Hi ha fotogrames on realment no es veu alguna extremitat (la del darrere surt del pla i la del davant a vegades queda amagada darrere la del darrere o entre el pelatge). Quan les potes no es veuen també s'ha comptat com a no detectada però, degut a que és impossible detectar una pota que no es veu, el percentatge d'encerts hauria de sortir més elevat.

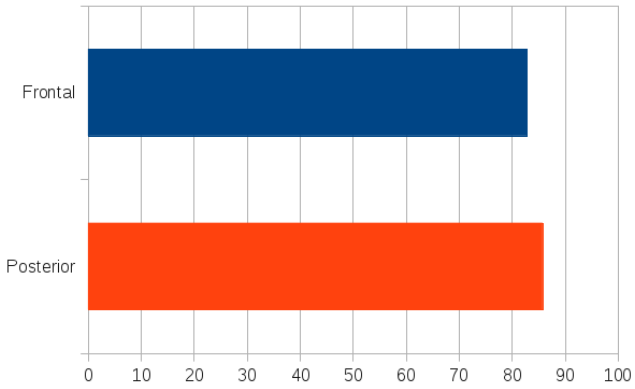


Figura 6.21: Percentatges d'encerts - *Vid32*

- *final_Vid36.avi*:
  - Percentatge d'encerts (extremitat posterior): 86%
  - Percentatge d'encerts (extremitat frontal): 83%
  - Fotogrames analitzats: 479

  El mateix es pot dir d'aquest vídeo, el percentatge d'encerts és molt elevat però el redueix una mica el fet que el alguns fotogrames les potes simplement no es veuen. De totes maneres, un percentatge de detecció molt elevat.

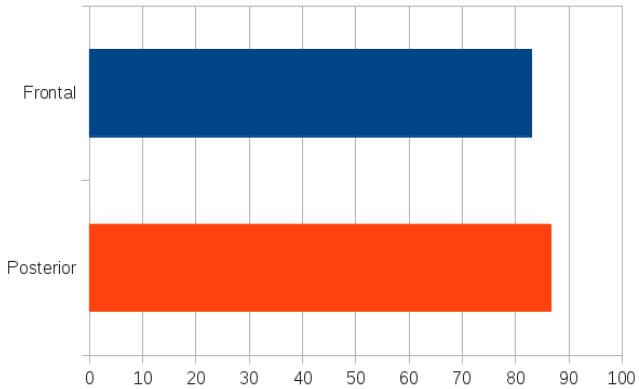


Figura 6.22: Percentatges d'encerts - *Vid36*

- *final_Vid44.avi*:
  - Percentatge d'encerts (extremitat posterior): 84%
  - Percentatge d'encerts (extremitat frontal): 77%
  - Fotogrames analitzats: 479

Aquest vídeo té les mateixes característiques que els dos últims però la pota del darrere surt més sovint del pla o està més estona aprop del límit esquerre de l'imatge, on la detecció falla més.

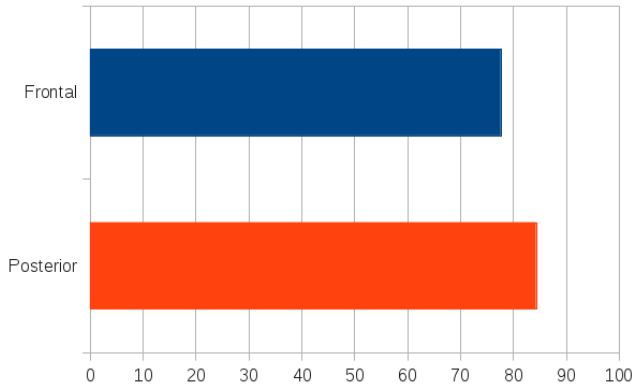


Figura 6.23: Percentatges d'encerts - *Vid44*

- *final_Vid64.avi*:
  - Percentatge d'encerts (extremitat posterior): 52%
  - Percentatge d'encerts (extremitat frontal): 72%
  - Fotogrames analitzats: 479

  Aquest vídeo és bastant dolent de per si degut a que les extremitats no es veuen en molts dels fotogrames (fora d'escena, amagades sota el pelatge, etc.). Aquest fet comporta que el percentatge de detecció baixi molt.

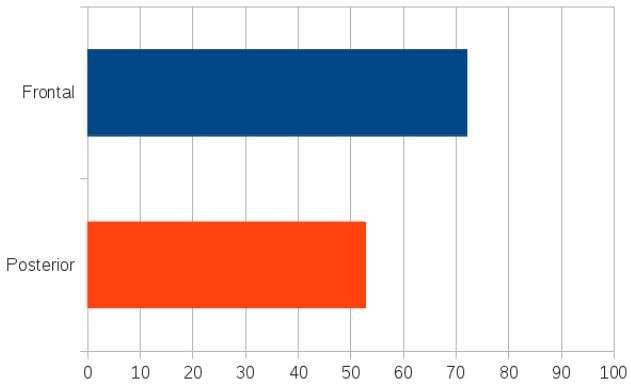


Figura 6.24: Percentatges d'encerts - *Vid64*

- *final_Vid81.avi*:

- Percentatge d'encerts (extremitat posterior): 80%
- Percentatge d'encerts (extremitat frontal): 73%
- Fotogrames analitzats: 479

Finalment, aquest vídeo també disposa de percentatges molt elevats però una altra vegada s'ha de tenir en compte que les extremitats no es veuen en varis fotogrames i això fa que el percentatge de detecció baixi.

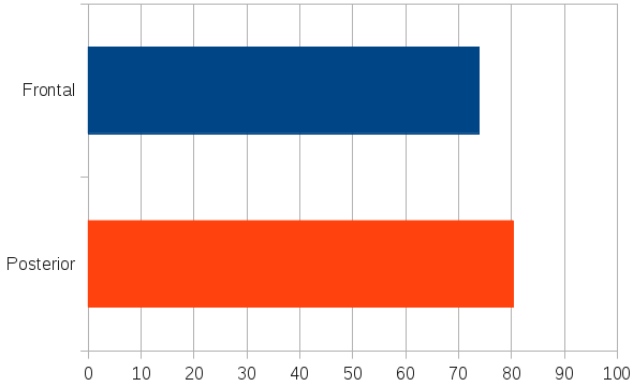


Figura 6.25: Percentatges d'encerts - *Vid81*

Un dels problemes amb aquestes estadístiques és, doncs, que es considera que en escena sempre hi ha visible una pota del davant i una altra del darrere quan, en realitat, a vegades es troben fora d'escena o amagades entre el pelatge o fins i tot una pot amagar l'altra. Així doncs, els percentatges d'encert serien una mica més elevats del que mostren les estadístiques anteriors si es consideressin com a negatius només els casos que l'extremitat es pot veure però no es detecta.

De totes maneres, el percentatge d'encert mitjà pel que fa a les potes posteriors surt del 77.875% i el de les potes frontals surt del 76.5%, molt igualats i superiors al 75%. Així doncs, es pot considerar que el procediment encerta la posició de la pota tres de cada quatre fotogrames.

Capítol 7

Conclusions

En aquest capítol es parla sobre l'assoliment dels objectius plantejats a la Secció 1.2, així com el compliment de l'abast exposat a la Secció 1.4, realitzant una anàlisi de les possibles variacions que han succeït respecte la planificació original. A més, es duu a terme un comentari dels resultats obtinguts a la Secció anterior i finalment es comenta l'aportació que proporciona l'eina creada per al grup de recerca del *Neuroscience Institute* de la Universitat de Princeton i pels seus investigadors en general.

Així doncs, el que s'ha aconseguit desenvolupar en aquest projecte és una eina multi plataforma capaç de localitzar i realitzar el seguiment de les extremitats frontals i posteriors d'un ratolí a partir de vídeos enregistrats en un entorn controlat de proves de les característiques mencionades a la Secció 1.3 utilitzant algorismes de visió i intel·ligència artificial. Més concretament i mencionant directament els objectius marcats a l'inici del projecte, s'ha aconseguit segmentar el ratolí distingint-lo de la resta de l'entorn, s'ha aconseguit localitzar amb percentatges elevats d'encert les extremitats visibles de l'animal i fer-ne el seguiment, s'ha aconseguit extreure'n mesures valuoses (el posicionament en cada fotograma, dipositat en arxius de sortida), s'ha aconseguit avaluar l'èxit de la detecció per cada fotograma dels vídeos proporcionats (utilitzant l'eina de validació de la que es fa menció a la Secció 6.8) i, en conjunt, s'acabarà proporcionant una eina a l'equip de recerca que els pot arribar a resultar útil. Per aquesta mateixa raó, es pot considerar que s'han assolit els objectius i s'ha acomplert l'abast marcat amb escreix.

Al llarg del projecte no han variat els objectius però sí que han anat evolucionant els camins per arribar a assolir-los. S'han provat molts més algorismes (molts d'ells mencionats al Capítol 4) que no pas els que han acabat formant part de la solució final com, per exemple, altres mètodes de segmentació o mètodes per localitzar el cap de l'animal que al final no s'han acabat utilitzant. Algunes de les idees que no s'han acabat integrant formen part de possibles millores o variacions que es deixen com a possible treball futur, mencionades al Capítol 8.

Pel que fa al seu desplegament i execució, al tractar-se d'un *software* desenvolupat en *Python*, es pot realitzar amb el mínim esforç, considerant que es disposa dels requeriments mínims en forma de llibreries de les que es parla a la Secció 1.5 i utilitzant la informació continguda dins les instruccions d'ús que es poden trobar a la Secció 2.4 de l'Annex.

Pel que fa a l'apartat tècnic de l'aplicació, s'ha realitzat un model de classes que permet el seu fàcil manteniment i extensió degut al baix acoblament entre els objectes, explicat pel fet

d'haver respectat els patrons *GRASP* en l'anàlisi i disseny realitzat al Capítol 5. La inclusió del patró *chain of responsibility* permet afegir funcionalitat de forma ràpida i lligar-la amb la resta d'algorismes de processat de forma senzilla. A més, el codi s'ha documentat utilitzant notació *Sphinx* en anglès de tal manera que els investigadors del *Neuroscience Institute* puguin entendre què fa cada classe i mètode de forma ràpida.

S'ha comentat que l'eina proporciona un percentatge elevat d'encerts pel que fa a la localització d'extremitats. Això queda demostrat amb els vídeos generats i amb la validació realitzada amb els vuit vídeos dels que es disposava que respectaven les condicions descrites a la Secció 1.3, resultats de les quals queden resumits a la taula següent:

Encerts	*Vid1*	*Vid10*	*Vid14*	*Vid32*	*Vid36*	*Vid44*	*Vid64*	*Vid81*	**Mitjana**
Posterior	73%	81%	71%	85%	86%	84%	52%	80%	**77.875%**
Frontal	80%	78%	78%	82%	83%	77%	72%	73%	**76.5%**

És important notar que els percentatges obtinguts consideren que cada extremitat és present en tots els fotogrames. Això vol dir que s'han comptat com a fallades les deteccions en fotogrames on realment no es pot detectar res ja sigui perquè l'extremitat queda fora de l'imatge o perquè es troba encoberta entre el pelatge o darrere l'altra pota.

Pel que fa a l'aportació real que proporciona el *software* desenvolupat en aquest projecte pel grup de recerca, essent realistes es podria dir que els pot ser d'utilitat com a fonament per a conèixer el posicionament de les dues potes de forma automàtica tal i com requereixen. Els pot resultar útil en el seu estat actual però es considera que els pot resultar encara més útil com a base per a construir un *software* més complexe que s'adapti encara més a les seves necessitats, casos d'ús o maneres de treballar ja sigui mitjançant la integració d'una interfície gràfica, reforçant, afegint o canviant algorismes, etcètera. Per a més informació, al Capítol 8 s'introdueixen certes consideracions i millores deixant-les com a possible treball futur o possibles línies noves d'investigació.

Capítol 8

Treball futur

Com resulta evident en un projecte d'aquestes característiques, sempre es poden anar afegint millores i nova funcionalitat. Hi ha molta feina que es pot fer més enllà dels objectius marcats originalment i a continuació se n'exposa una llista breu:

- Millores en el rendiment: un dels requeriments inicials era disposar d'un entorn de proves àgil i dinàmic i, entre altres raons, això va comportar que el llenguatge de programació escollit per a dur a terme el desenvolupament fos *Python*. Aquest llenguatge de programació té moltes virtuts però l'alt rendiment no n'és una d'elles. Això comporta que els diversos algorismes de visió artificial que s'han implementat directament en *Python* siguin més ineficients i lents que no pas si s'haguessin implementat utilitzant altres llenguatges com *C++*.

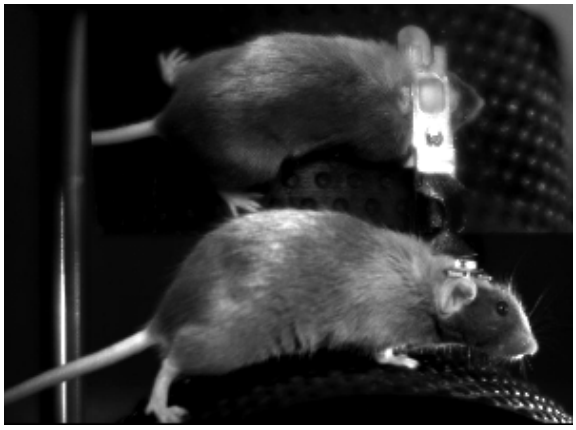
  Els algorismes de les llibreries *OpenCV* estan implementats en *C/C++* i la interfície *Python* simplement actua de *wrapper*, fent que les funcions en si siguin més eficients i ràpides. No obstant, alguns algorismes s'han implementat directament en *Python*, com seria el cas dels de la construcció de l'esquelet topològic, i aquests sí que són notòriament més lents. Una alternativa proposada seria implementar aquests algorismes directament en *C++* i fer *bindings* per *Python* per executar-los dins el codi actual.

  Una altra alternativa seria, ara que ja es disposa de la solució final i ja no cal disposar d'un entorn de proves tan dinàmic i flexible, implementar tota l'aplicació directament en *C++*. Això comportaria un augment dràstic en el rendiment i és una opció a tenir en compte si el temps d'execució acaba esdevenint un problema. Aquesta millora s'obtindria a costa de perdre l'actual execució multi plataforma i caldria disposar de compilacions diferents per cada sistema operatiu al que es volgués donar suport.

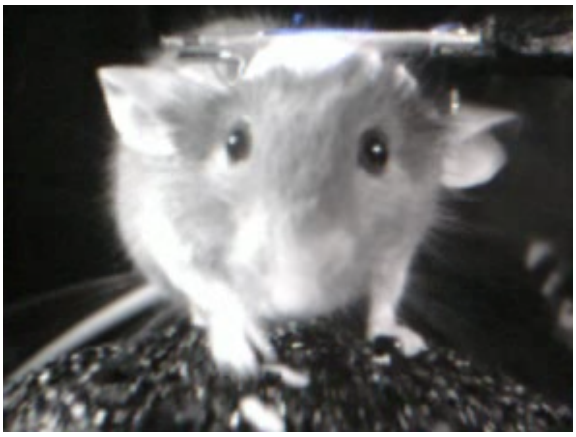
- Interfície gràfica: la solució entregada cal esser utilitzada a partir de la línia de comandes i només es pot consultar el resultat un cop ha finalitzat el processat de tot el vídeo. Una millora orientada a l'equip del *Neuroscience Institute* per facilitar-los la feina podria ser l'embolcallament de tota la lògica del programa al voltant d'una interfície gràfica que permetés anar visualitzant el procediment en temps real, avançar fotograma a fotograma, canviar paràmetres en calent, etc.

- Processat d'altres punts de vista: l'equip del *Neuroscience Institute* disposa d'un nou sistema de captura d'imatges dins l'entorn de prova que proporciona diferents punts de vista dins d'aquest (captura frontal, lateral i en picat del ratolí durant els experiments). En concret, cap a la finalització del treball es van rebre nous vídeos en molt alta definició

on caldria experimentar per poder realitzar el *tracking* tal i com s'ha fet amb els vídeos de prova dels que es disposava al principi.



(a) Punt de vista lateral i en picat



(b) Punt de vista frontal

Figura 8.1: Altres punts de vista

- Utilitzar la informació de la posició del cap: la posició del cap ja es calcula durant el procediment de classificació però aquesta informació finalment no s'ha acabat utilitzant per a res. L'idea inicial era utilitzar la posició del cap com a punt de referència per intentar determinar la posició de les potes però el procediment que s'ha acabat utilitzant ha fet que això ja no fes falta.

- Diferenciar la pota frontal de la pota posterior: el procediment actual és capaç de trobar la posició de les extremitats però en cap moment es diferencia si es tracta de la pota del davant o la del darrere. Una possible aproximació podria ser utilitzar la informació de posició del cap per a determinar quina pota és la del davant (la més propera al cap) i quina la del darrere. Això és una mica més complexe del que sembla, però, ja que a vegades es classifiquen tres potes (en algun vídeo es poden arribar a apreciar tres extremitats en alguns moments) o a vegades només se'n troba una de sola.

- Entrenar els classificadors *Haar* amb més exemples d'altres vídeos amb altres perspectives: el classificador d'extremitats entrenat funciona molt bé per tots els vídeos de prova dels que es disposava però caldria veure si es comportaria així de bé amb vídeos des d'altres perspectives o amb altres característiques. Sempre es pot intentar entrenar un nou classificador amb molts més exemples positius i negatius si s'ha d'acabar treballant amb vídeos que no tinguin les característiques dels que s'ha provat.

- Utilitzar la informació de continuïtat: per a reforçar el procés de segmentació i classificació es podria utilitzar la informació trobada en fotogrames anteriors. D'aquesta manera es podrien descartar falsos positius i, fins i tot, marcar una extremitat en un fotograma on realment no s'ha pogut trobar res.

Bibliografia

[1] Numpy. http://www.numpy.org/.

[2] Opencv - open source computer vision. http://opencv.org/.

[3] Python programming language. https://www.python.org/.

[4] scikit-image - image processing in python. http://scikit-image.org/.

[5] Scipy. http://www.scipy.org/.

[6] Sphinx - python documentation generator. http://sphinx-doc.org/.

[7] Kobus Barnard Deva Ramanan, D.A. Forsyth. Detecting, localizing and recovering kinematics of textured animals.

[8] Jitendra Malik Jianbo Shi. Normalized cuts and image segmentation. august 2000.

[9] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut -interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (SIGGRAPH)*, August 2004.

[10] Avi Mehta Suman Tatiraju. *Image Segmentation using k-means clustering, EM and Normalized Cuts.*

[11] C.Y. Suen T.Y. Zhang. A fast parallel algorithm for thinning digital patterns.

[12] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[13] Ramin Zabih Yuri Boykov, Olga Veksler. Fast approximate energy minimization via graph cuts.

[14] Richard Hall Zicheng Guo. Parallel thinning with two sub-iteration algorithms.