Universitat Oberta de Catalunya

uoc.edu

Sistema de visió artificial per a la detecció de fuites de fluids

Autor: Marc Segura Valls
Grau de Tecnologies de la Telecomunicació
Aplicacions multimèdia basades en processament de la senyal

Consultora: Lourdes Meler Corretjé
Professor responsable: David García Solórzano

Girona, 9 de gener de 2019

FITXA DEL TREBALL FINAL

Títol del treball:	*Sistema de visió artificial per a la detecció de fuites de fluids*
Nom de l'autor:	*Marc Segura Valls*
Nom del consultor/a:	*Lourdes Meler Corretjé*
Nom del PRA:	*David García Solórzano*
Data de lliurament (mm/aaaa):	*01/2019*
Titulació o programa:	Grau de Tecnologies de la Telecomunicació
Àrea del Treball Final:	*Aplicacions multimèdia basades en processament de la senyal*
Idioma del treball:	*Català*
Paraules clau	*Processament d'imatges, OpenCV, Detecció de fuites.* *Image Processing, OpenCV, Leak detection.*

Resum del Treball

El treball presentat en aquest document, tracta sobre de creació d'un sistema que té com a finalitat principal, detectar fuites de fluids en qualsevol tipus d'instal·lació. El projecte, es desenvolupa dins l'àmbit del processament digital de la imatge, més concretament, el de la visió artificial. Els elements bàsics d'aquest sistema, són una càmera d'alta qualitat i una aplicació que obté la imatge en temps real, la processa per tal de poder analitzar-la i prendre una decisió.

La implementació està feta amb el llenguatge de programació C++ sobre la plataforma .NET de Microsoft, i amb la llibreria externa de visió artificial OpenCV.

Utilitza diverses tècniques de la visió artificial, i ofereix tres modes d'operació, per tal d'oferir més versatilitat, a més, disposar de més informació per tal d'obtenir uns conclusions més sòlides i resoldre quin és el més efectiu i fiable.

S'implementen vàries de les funcions més bàsiques de la visió artificial com captura d'imatge a temps real, conversió de RGB a escala de grisos, carregar, guardar i mostrar imatges, etc. Però també, diverses operacions i transformacions com suavitzat Gaussià, algoritme de detecció de contorns *Canny*, subtracció entre imatges, operacions morfològiques, binarització, recerca de contorns, superposició de zones, etc.

El primer mode es basa en la de detecció de contorns, el segon i el tercer, en la subtracció entre imatges. Aquestes dues difereixen en què una té una imatge mestre fixa per fer la subtracció amb la real, i l'altre, aquesta imatge mestre es va actualitzant periòdicament. Amb la subtracció entre les dues imatges, s'obté una imatge resultant on només hi apareixen els nous elements.

Els resultats són satisfactoris, i el tercer mode, on la imatge mestre es va actualitzant automàticament, és el més robust i fiable, el que supera més obstacles i pertorbacions.

És un sistema creat per créixer i adaptar-se a noves tecnologies. Ara mateix, només es pot configurar dues zones de vigilància, però en una pròxima versió i amb una càmera servocontrolada podrà vigilar un gran nombre de punts.

Abstract:

This project aims to control the fluid leaking detection on domestic and industrial installations, indoors and outdoors. The detection is done by camera capturing and image processing.

Software is built with C++ language and the library of computer vision called OpenCV. The integrated development environment (IDE) used, is the Microsoft Visual studio that runs on the .NET framework.

The detection can be done by three different methods. For all the options, the user previously has to configure the watching zones areas (this version allows up to two zones). Also, there are specific settings for each one. Once, a leaking is detected, the user is notified by email.

The system uses some of the most common functions and operations of computer vision, such as color spaces conversion, smoothing, edge detection, morphology, threshold, subtraction, contour finding, drawing, etc.

Índex

Llista de figures

1. Introducció

1.1 Context i justificació del Treball

La idea d'aquest treball, sorgeix de la necessitat de portar una solució a un problema bastant habitual en les instal·lacions industrials, les fuites de fluids. Per tant, la finalitat principal d'aquest projecte és la de crear un sistema de baix cost capaç de detectar en temps real una possible fuita de qualsevol tipus de fluid, en diferents instal·lacions. Està focalitzat sobretot en el món industrial, però també se'n podrà treure profit a nivell domèstic.

Analitzant les tecnologies més actuals, trobem, que la visió artificial (o visió per computador) pot ser una bona solució per assolir aquesta detecció autònoma en temps real, perquè és un camp ja molt madur i ofereix un gran ventall d'eines, moltes d'elles de programari lliure. També, es pot trobar una càmera d'alta qualitat a un preu popular.

En un entorn industrial es pot arribar a treballar amb un gran ventall de fluids: oli tèrmic, oli hidràulic, oli lubricant, coles, tintes, concentrats, aigua, etc; tant a les instal·lacions d'una nau industrial, com també a la maquinària de les línies de producció. La majoria d'aquests fluids solen ser molt cars, i en alguns casos perillosos o contaminants. Per tant, una fuita, per exemple en una junta entre mànegues, o en una vàlvula, pot suposar un cost extra innecessari; tant per el malbaratament del producte com per el cost que comporta netejar-ho. A més, la majoria de plantes treballen 24 hores i 7 dies la setmana, amb el punt en contra que durant les nits i els caps de setmana hi ha la meitat de personal, cosa que fa que el grau de vigilància sigui menor durant vàries hores. En el sector domèstic el més susceptible seria el de la instal·lació d'aigua, tant de la xarxa com el de la caldera. També existeixen les calderes d'oli tèrmic.

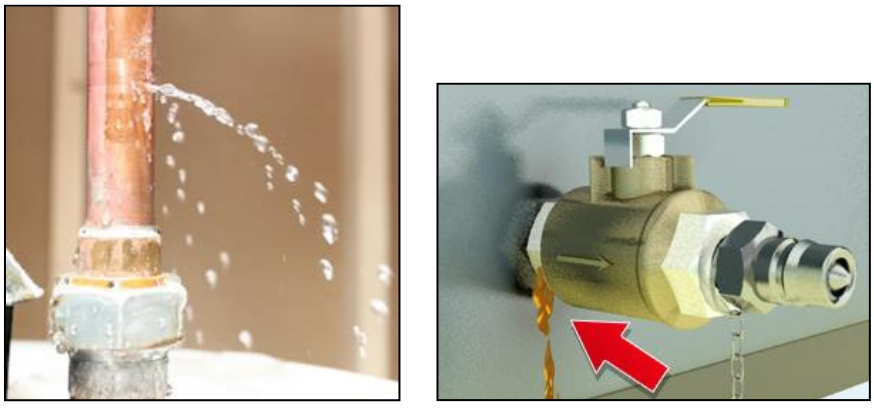

Figura 1. Exemple de diferents tipus de fuites de fluids en instal·lacions.

Fins a l'actualitat, la detecció de fuites s'ha fet mitjançant la supervisió humana (no molt efectiva per raons òbvies), la utilització de sensors de nivell, sensors de pressió, bàscules, etc. Si es detecta, per exemple una baixada de nivell en el dipòsit del producte inusual o bé una disminució de pressió dins la instal·lació, es llença una alarma i es procedeix a la intervenció correctiva. Aquesta supervisió per sensors, en alguns casos, pot donar l'avís quan ja és massa tard i s'ha perdut molt de fluid. Per tant, seria necessari un sistema en temps real.

A dia d'avui, ja existeix alguna solució de visió per computador i amb càmera d'infraroja per detectar el canvi de temperatura, però són extremadament cars.

Per acabar, es reserva una part per dur a terme un experiment per un cas particular i molt comú en les màquines industrials d'impressió per flexografia [1]. A la part on es transfereix la tinta a cada unitat, arrel del desgast d'unes peces ubicades en els laterals de la cambra del fluid, es produeixen unes fuites de tinta indesitjables, també al llarg d'aquesta, i degut al desgast de la fulla metàl·lica que ajuda a mantenir la tinta dins la cambra. Detectar-les a temps significaria un gran estalvi dins el cost de la producció.

Aquest cas, no s'ha pogut dur a terme en les proves experimentals d'aquest treball, degut a no tenir encara l'accés a una planta d'impressió. Però tant aviat com es pugui, es planificarà una visita en unes instal·lacions reals per fer les proves.

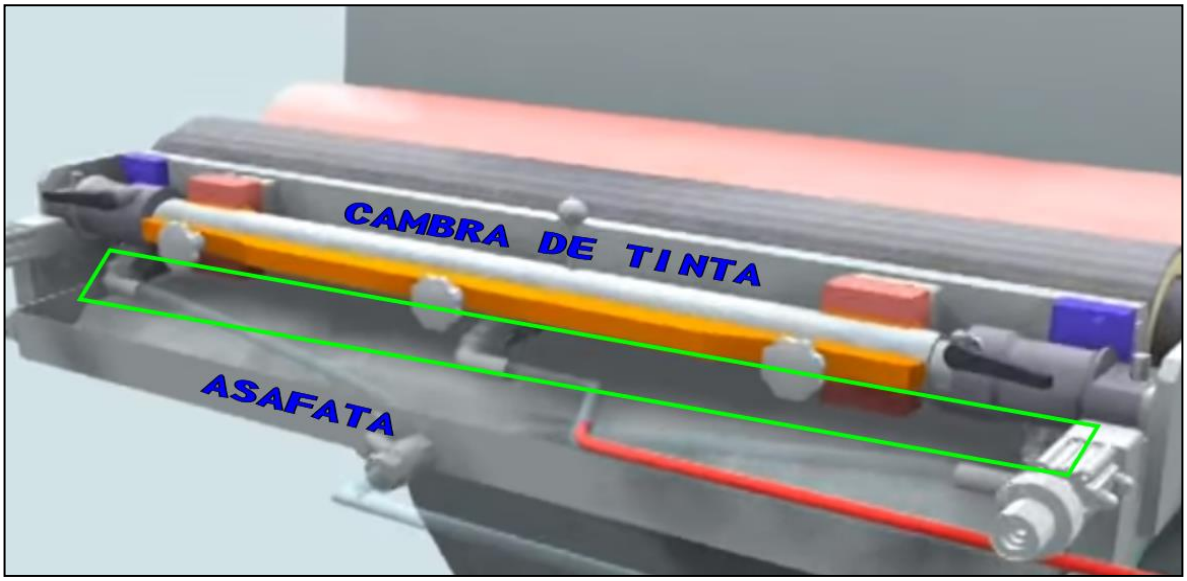

Figura 2. Unitat de transferència de tinta d'una màquina flexogràfica. En verd la zona susceptible a fuites.

1.2 Objectius del Treball

Mitjançant un IDE (Integrated Development Environment) versàtil, un llenguatge de programació potent, les llibreries adients, i una càmera de vídeo, es dissenya i s'implementa un sistema detector de fuites de fluids capaç de notificar a un sistema central o un usuari concret. Un cop assolida aquesta fita, es podria passar a una segona fase (en un futur molt proper) en el qual es robotitzaria la càmera per tal de fer un rastreig a tots els punts memoritzats i analitzar-los un a un de manera cíclica. També es pot pensar en una funció especial per càmera de visió nocturna, i així resoldre l'anàlisi amb falta de llum. Com a culminació, el dia que el preu d'una càmera infraroja sigui molt més assequible, es podrà tenir un sistema altament fiable, i a més detectar gasos.

Els objectius principals del projecte són:

- Dissenyar i implementar una aplicació de visió artificial destinada a detectar fuites de fluids de tot tipus de productes.
- Escollir la plataforma de programació i la llibreria que s'ajusti millor a les necessitats del projecte.
- Elecció d'una càmera de vídeo amb una qualitat suficient per a poder visualitzar fuites de tot tipus de productes, a més, d'un preu raonable.
- Crear una interfície fàcil de configurar i utilitzar per l'usuari. Que sigui molt intuïtiva.
- L'aplicació ha de ser escalable i oberta a que si puguin afegir noves funcionalitats de manera còmoda i ràpida.
- L'aplicació utilitzarà més d'una tècnica de detecció, per tal de poder comparar i obtenir quina ofereix els millors resultats.
- L'usuari configurarà els punts a analitzar (a la primera versió es podran configurar dues zones de vigilància) i el sistema li enviarà notificacions per correu electrònic en cas de fuita.
- Testejar l'aplicació amb el màxim tipus de productes: oli lubricant, oli tèrmic, tinta, cola, aigua...
- Controlar els moviments i el zoom de la càmera de manera remota.

1.3 Enfocament i mètode seguit

Des del moment que va sorgir la idea d'aquest treball, es va tenir molt clar que havia de ser un producte nou, i totalment desenvolupat des de zero. Existeixen diverses productes que permeten implementar de manera senzilla funcions de visió per computador. És el cas de l'extensió de Matlab [2] anomenada *computer vision System toolbox*. Però Matlab és una eina de programari matemàtic destinat a tasques de laboratori, i no per la creació de nous productes.

Es considera, que com que a part d'experimentar, també es vol crear una solució que pugui tenir un lloc al mercat, s'ha de crear una aplicació amb una plataforma de programació popular dins el sector. Llavors, seleccionar una llibreria de visió per computador que es pugui utilitzar amb aquesta plataforma.

A part de treballar amb les principals funcions de la visió artificial, també es vol tenir l'experiència de dissenyar un producte de caràcter professional, i amb un disseny que pugui tenir una bona acceptació per a usuaris que no tinguin grans coneixements en computadors. Per tant, que tota la part més tècnica sigui totalment transparent per aquests possibles clients.

1.4 Planificació del Treball

El punt de partida per la realització d'aquest projecte, ha estat haver cursat l'assignatura processament de la imatge, i que dona una molt bona base. A partir d'aquí, s'han consultat varis articles, tesis, altres treballs de final de carrera, pàgines web, etc. El resultat d'aquesta recerca es mostra a l'apartat sobre l'Estat de l'Art. Molts d'aquests recursos es van citant durant la memòria, i es llisten a l'apartat de bibliografia.

S'ha fet una planificació temporal inicial de les tasques a realitzar en forma de diagrama de Gantt. Aquesta planificació, al principi era més orientativa, donat que no es disposava d'informació suficient. Un cop assolit l'Estat de l'Art, s'han reajustat algunes tasques. Les fites són les que venen marcades per el pla docent. A la pàgina següent, a la figura 3, es mostra aquest diagrama de Gantt.

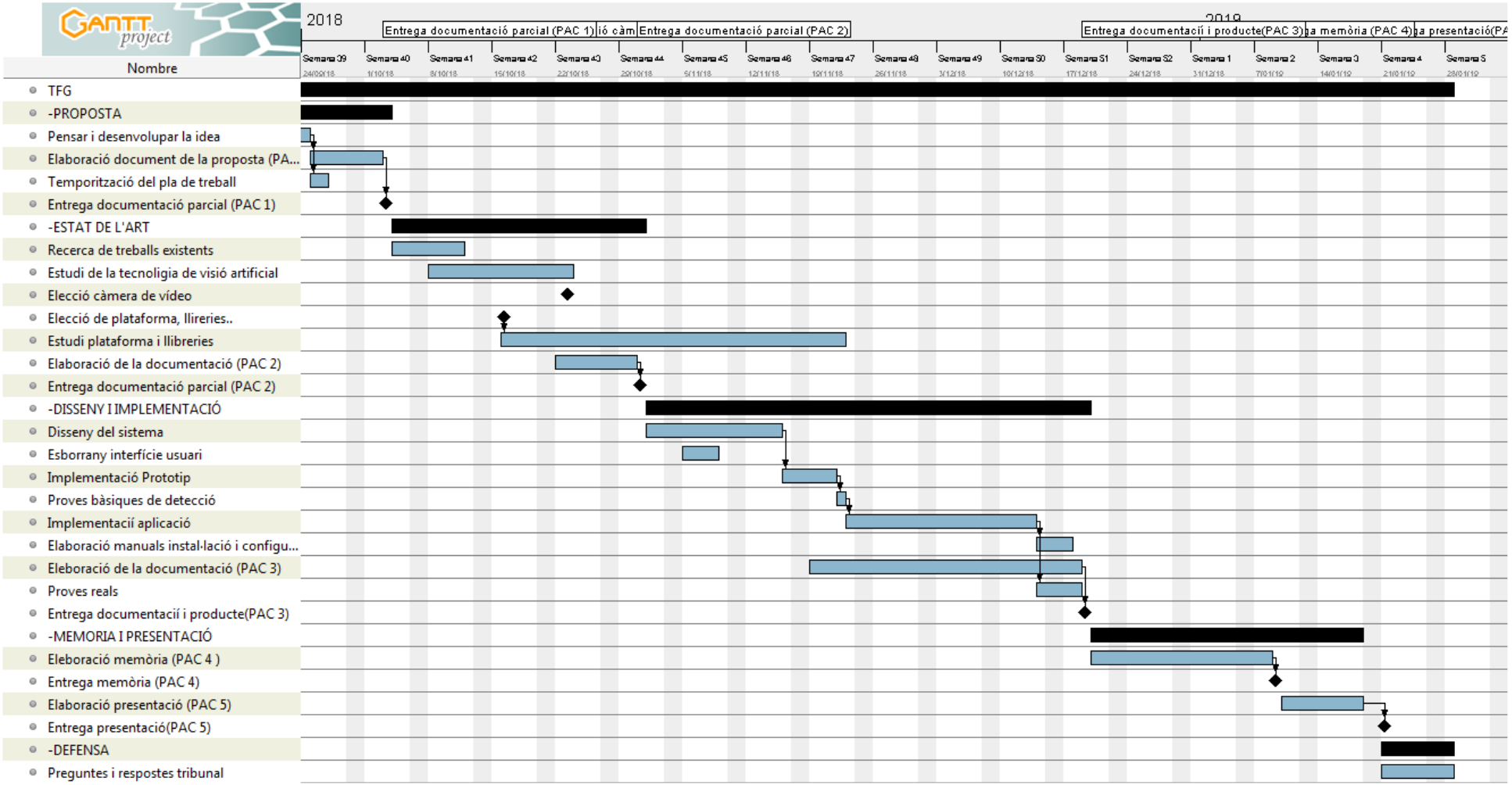

Figura 3. Diagrama de Gantt de la planificació del treball.

Per realitzar la implementació d'aquest sistema, s'ha utilitzat el llenguatge de programació C++ amb l'IDE i compilador Microsoft Visual Studio. S'ha utilitzat la llibreria de visió artificial OpenCV.

La captura de la imatge a temps real es realitza amb càmera de Logitech C920 HD USB Pro.

1.5 Breu sumari de productes obtinguts

Un cop acabat el projecte, els productes obtinguts són: un projecte de Microsoft Visual Studio 2017 C++ amb tots els fitxers necessaris per el desenvolupament, un fitxer que és el programa compilat (l'acompanyen altres fitxers de suport com per exemples la icona, imatges, etc.) executable per entregar a l'usuari i la llibreria OpenCV versió 4.3.3.

A part, aquesta memòria (PAC 4), on a part de explicar en detall totes les fases del projecte, també explica com configurar l'entorn de desenvolupament amb la llibreria OpenCV.

1.6 Breu descripció dels altres capítols de la memòria

La base d'aquesta memòria és el recull de tota la documentació que s'ha elaborat per les entregues parcials del treball. Llavors, l'estructura de la memòria ve marcada per aquests treballs previs. A més, si afegeixen les tasques finals.

Aquestes entregues han estat les següents:

- PAC 1. La proposta del projecte.
- PAC 2. L'Estat de l'Art.
- PAC 3. Disseny i Implementació.

A partir d'aquesta informació podem obtenir tots els capítols de la memòria, i que són els que es detallen a continuació:

- Capítol 1. *Introducció:* Si exposa la motivació i les necessitats del treball, els objectius principals, el mètode seguit, la planificació i els productes finals obtinguts.

- Capítol 2. *Estat de l'Art*: És un estudi exhaustiu sobre l'estat actual, tant dels sistemes de detecció de fuites de fluids, com de la visió artificial. Aquesta part és la que ajuda a desxifrar quin camí seguir per aconseguir assolir les expectatives.

- Capítol 3. *Disseny*: És on es representa el diagrama de blocs del sistema, i a partir d'aquí, es desgrana per poder dissenyar la interfície gràfica i les principals funcions de l'aplicació.

- Capítol 4. *Implementació*: Explica en detall el funcionament del programa i les funcions que s'han creat. També quines de les funcions de la llibreria de visió per computador que s'han utilitzat.

- Capítol 5. *Proves experimentals i resultats*: És un informe sobre les proves que s'han fet sobre una simulació d'un entorn industrial amb tots els modes de funcionament.

- Capítol 6. *Conclusions i línies futures*: Descriu la valoració personal del treball i de les eines utilitzades. A més presenta les línies de futur que el projecte ha de seguir a les properes fases.

2. Estat de l'Art

Un cop ja definit el propòsit del projecte, és hora de conèixer quina és la situació actual i com s'estan tractant en el mercat actual els sistemes de detecció de fluids. S'exposa una visió general de les diferents solucions i mètodes que existeixen fins al moment. D'aquesta manera es tindrà una percepció més real sobre la repercussió que pot tenir el treball un cop convertit en producte. I el més important, saber si alguns dels productes ja existents es basen en la visió artificial.

Seguidament ha fet falta estudiar quines són les llibreries més populars de visió artificial susceptibles a ser seleccionades per el treball proposat. També, conèixer amb quins llenguatges de programació es poden utilitzar. D'aquesta manera, en funció de diferents criteris, com que siguin tecnologies madures, eficients, documentades... s'ha fet una selecció més efectiva.

2.1 Sistemes de detecció de fuites

La detecció de fluids i líquids en el món actual, apareix arrel de diferents possibles riscos que poden succeir en diferents entorns: industria del petroli, químiques, centres de servidors de dades, etc. En les fuites de productes inflamables, existeix el risc d'incendis i explosions; en els químics, contaminació ambiental i en alguns casos danys irreversibles al personal. En el cas de les sales plenes de servidors on es tracten dades tant importants com per exemple: sistemes de transmissió bancària, centres de control aeri, de tràfic, etc. una fuita d'aigua no detectada, pot tenir conseqüències molt perjudicials i a gran escala.

A més, els sistemes de detecció de fuites ajuden a millorar la productivitat i la confiabilitat, ja que assegura una reducció d'inactivitat i tasques d'inspecció i manteniment.

Existeix la norma creada als Estats Units per l'Institut americà del petroli, i registrada com a API 1130 que es centra en el disseny, proves i operacions de sistemes CPM (Computational Pipeline Monitoring) [3]. Utilitzen un algoritme d'aproximació que ajuda a detectar anomalies hidràuliques en una instal·lació de canonades. Aquesta norma divideix els sistemes de detecció de fuites entre els basats en la ubicació interna i externa, i cadascun utilitza tècniques diferents en funció del tipus de producte i instal·lació.

Les tècniques de detecció interna inclouen [4] [5]:

- Proves hidrostàtiques amb recipients a pressió.
- Sistemes digitals amb cables de detecció instal·lats i alineats sota els conductes.

- Canvis en la lectura, complexes algoritmes i models matemàtics mitjançant sensors de flux, pressió o temperatura.

Les tècniques de detecció externa inclouen [4] [5]:

- Càmeres tèrmiques amb programari d'anàlisi de vídeo.
- Infrarojos (ideal per instal·lacions soterrades).
- Cables de fibra òptica i el seu canvi de la reflexió del pols, instal·lats al llarg dels conductes.
- Sensors de vapor.
- Detecció d'emissió acústica que genera un líquid al passar per un orifici.
- Sistemes digitals amb cables de detecció (igual que en la detecció interna).
- Mètodes biològics on s'utilitzen gossos entrenats i el seu gran olfacte.

La rèplica europea és la norma creada a Alemania TRFL (Technische Regel für Fernleitungsanlagen) [5]. Aquesta, és més extensa, ja que cobreix també conductes que transporten líquids inflamables i perillosos. També els que transporten gas. Aquesta norma separa les deteccions en 5 diferents tipus.

Funcions requerides per la detecció de fuites [5] [6]:

- En continua, que es dona durant el funcionament estable.
- Durant operacions transitòries.
- Durant operacions de tancament.
- Fuites progressives.
- Localització ràpida.

Molts dels sistemes actuals requereixen molta instrumentació, i els costos i complexitats de les instal·lacions són molt alts. Aquests sistemes, per tant, queden limitats a àrees d'alt risc. La inversió que tindrà el nostre treball serà molt menor, ja que l'únic dispositiu és una càmera d'alta qualitat que a dia d'avui no requereix una gran inversió. En entorns d'alt risc podria servir com a complement, i a instal·lacions més comunes podria ser una solució final.

Entre les diferents solucions actuals, se'n troba una que fa anàlisi d'imatges, en aquest cas a partir imatges captades amb una càmera tèrmica. Cal estudiar-la doncs, més a fons per tal de saber quins punts pot tenir en comú amb aquest treball. Per això cal dedicar-hi un apartat.

Com que la nostra aplicació pretén ser útil i trobar un lloc en el mercat, s'haurà de seguir aquestes normes. A més, estar atens, per les noves que puguin aparèixer en el futur. Un cop el projecte estigui alliberat, periòdicament es revisaran les dues normes per saber si han publicat noves actualitzacions.

2.1.1 Detecció de fuites amb imatges tèrmiques

Les imatges tèrmiques avançades sotmeses a anàlisi de vídeo, actualment estan ajudant al camp de la detecció de fuites de fluids. L'empresa IntelliView [7] és una de les pioneres, i ho fa a partir d'una càmera dual coneguda com a DCAM. Aquesta càmera combina dos imatges: una captada per els sensors infrarojos i una captada per els sensors de color.

La part més important d'aquest sistema és el mòdul que s'encarrega d'analitzar les imatges [8]. Ho fa mitjançant un ampli ventall d'eines i tecnologies diferents de la visió per computador i la intel·ligència artificial. Inclou la detecció d'objectes d'interès prèviament classificats. També té filtratge ambiental, que s'encarrega de gestionar els efectes ambientals com ombres, enlluernaments, etc. i impactes climàtics com pluja, neu, boira, etc. Aquests filtres són de gran utilitat, donat que són causes que solen generar falsos avisos.

Aquest mòdul resideix en una terminal remota, on si poden veure imatges en temps real de diferents ubicacions, i a més, controlar les càmeres, administrar les regles d'alarma i fer totes les configuracions necessàries.

Al tractar-se d'una empresa privada i amb finalitat empresarial, no posen a disposició les llibreries, funcions i algoritmes que han implementat.

Els sensors infrarojos són de microbolomètre no refredat [7], i estan sorgint com un mètode nou i efectiu per visualitzar, detectar i generar alertes d'emissions de líquids hidrocarburs. Capten la radiació infraroja, que és un tipus de radiació electromagnètica que es denomina radiació tèrmica, generada per la calor. La clau d'aquesta tecnologia, és que capta la calor dels objectes sense necessitat d'il·luminació ambiental. Quan el producte que s'està vigilant accedeix a l'exterior arrel d'una fuita, es podrà distingir per la diferència de calor amb els elements del fons.

El procés per detectar una fuita real, pot tardar uns 30 segons. Actualment, s'estan començant a establir sobretot en instal·lacions de canonades aèries, refineries, mines, plantes químiques, plantes de tractament d'aigua [8].

Figura 4. Mostra del producte de IntelliView [8].

La part analítica del sistema és la més complexa, ja que una fuita pot tenir diferents comportaments com per exemple: degoteig, polvorització, raig, etc. També diferents atributs: temperatura tèrmica, mida, forma, etc. Això fa que constantment projectes com aquest, es mantinguin vius i es millorin les funcions actuals i se'n creïn de noves.

L'altre factor que determina la qualitat de detecció, és la distància en que es troba el punt a analitzar de la càmera, per això la càmera disposa d'una bona lent amb una gran capacitat d'augment, i a més, ofereix una alta resolució.

Aquest treball s'ha trobat amb obstacles molt similars. Encara que estigui enfocat a un altre destinatari i no incorpori la càmera tèrmica, per a la detecció de fuites de fluids en conductes en diferents tipus d'instal·lacions, els reptes són els mateixos. Però com que IntelliView és un sistema totalment tancat i només ells coneixen com s'ha fet el disseny i la implementació, el procés d'enginyeria ha seguit mètodes i tècniques diferents.

A més, en aquest projecte es pretén també fer deteccions per a cassos que surten del camp de les instal·lacions, com és el cas comentat al primer apartat sobre la cambra de transferència de tinta en una màquina industrial d'impressió flexogràfica. Aquest és un tant sols un exemple que s'ha exposat, per donar a entendre que hi han diferents situacions de fuites de fluids a part de les fuites en conductes convencionals.

Per tant, a mesura que el producte es vagi establint i el nombre d'usuaris vagi creixent, sorgiran nous casos.

2.2 La visió artificial

La visió artificial (o per computador) és una disciplina lligada a la intel·ligència artificial que s'encarrega de donar a un sistema autònom la capacitat d'adquirir, processar i analitzar imatges del món real, tal com pot fer l'ésser humà. Dedueix el món tridimensional a partir d'imatges bidimensionals processades, que són una matriu de píxels (unitat homogènia més petita d'una imatge digital). Utilitzar un sistema de visió artificial enlloc d'un sistema de visió biològic, en alguns casos pot suposar un avantatge, sobretot quan s'han de fer tasques repetitives o perilloses.

2.2.1 Visió biològica envers la visió artificial

La similitud entre un sistema de visió artificial i el sistema de visió humà és molt gran, on la càmera té les mateixa funcionalitats que l'ull. El que ja és més complicat, és que una màquina comprengui una imatge amb la mateixa facilitat que ho fa el cervell humà. La capacitat de detectar i diferenciar elements és una activitat que un sistema biològic fa de manera natural, i dotar a una màquina d'uns processos que siguin capaços d'igualar aquesta capacitat és un repte que encara té molt camí que recórrer. Però cal reconèixer que en l'actualitat, s'ha assolit ja, un gran nombre de propòsits gracies diverses tècniques, eines, algoritmes, models estadístics, models matemàtics, tecnologies, etc.

Així doncs, tal i com s'explica en [9], s'accepta que la visió humana és millor per la interpretació d'escenes complexes no estructurades, i és en les escenes estructurades on la visió artificial pot superar-la. Amb una bona precisió, resolució i una bona lent òptica a la càmera, un sistema pot inspeccionar una gran quantitat d'elements per minut que el sistema visual humà no és capaç, degut a que la càmera pot captar detalls molt més petits que l'ull, i processar la informació més ràpid. A més, la màquina pot ser més eficaç en treballs senzills i repetitius, que poden arribar a fatigar molt a una persona. També té un gran avantatge en la seguretat, ja que evita la participació humana en un procés o entorn, contaminant o perillós. Un altre factor important, és que la visió artificial pot treballar amb llum o sense.

2.2.2 Aplicacions actuals de la visió artificial

Cada vegada són més els sectors on si aplica aquesta disciplina. Primer, va ser a la indústria, sobretot per la inspecció automàtica. Després, ja s'ha anat afegint a varis nous camps, com la medicina, meteorologia, geologia, navegació, biologia, cartografia, construcció, robòtica de servei, agricultura, control de tràfic, seguretat, etc.

No entra dins l'abast d'aquest treball estudiar cascuna d'aquestes aplicacions o les més importants, però si conèixer on són els límits actuals d'aquest camp. Això ha ajudat a tenir més clar els objectius i fins on cal arribar en aquest projecte. Per exemple, avui dia existeixen sistemes amb uns algoritmes molt complexes capaços de detectar tot tipus d'objectes i perills a temps real, és el cas del sistema MagicEye [10], dissenyat per millorar la seguretat en la conducció, alertant al conductor de possibles riscos que poden aparèixer, i així evitar distraccions.

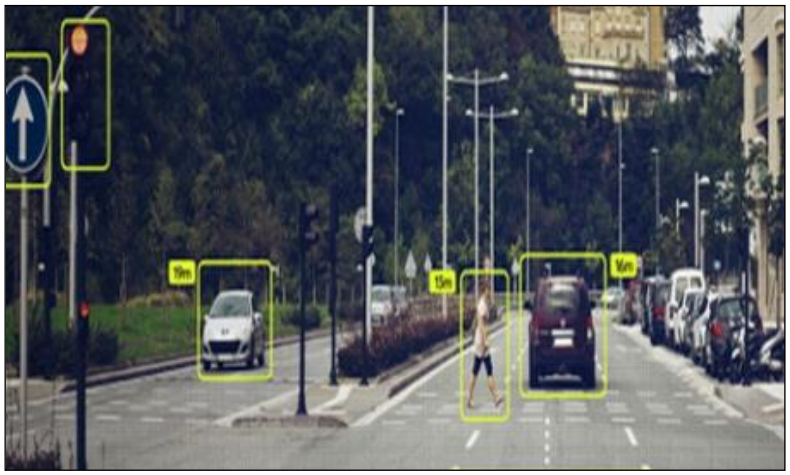

Figura 5. MagicEye, sistema que alerta al conductor sobre possibles riscos [10].

2.2.3 Processament d'imatges digitals

Per tal de poder extreure el major nombre de conclusions sobre una imatge, aquesta abans haurà de ser tractada, per obtenir la seva representació bidimensional en forma de matriu *M x N* (resolució espacial) de píxels. Cada píxel serà codificat per un conjunt de bits, que poden tenir diferents llargades en funció de la profunditat del color, i que va lligada amb la resolució d'amplitud $L = 2^k$ (on k és el número de bits per codificar cada píxel). Per exemple, amb *k = 8* tindríem 256 variacions de gris, i es considera una bona resolució.

Amb aquests paràmetres ja podem calcular la mida de la imatge *b*, que serà:

$$b = N \cdot M \cdot k(bits)$$

Interessa trobar, com s'explica a [11], l'equilibri entre la resolució espacial i la d'amplitud, ja que una imatge mostrejada i quantificada amb una baixa resolució espacial produirà l'efecte de pixelat, i en una amb la resolució d'amplitud baixa, hi apareixeran falsos contorns.

Les imatges a color solen utilitzar el model RGB, basat en els tres colors primaris, i es destina un byte per cada component. Cada component podrà agafar valors

entre 0 i 255, i en total representar 2^{24} = 16.777.216 colors. Hi han altres models com el HSV, HSL, CMY,... però les biblioteques més populars treballen amb el model RGB. També existeixen imatges de quatre canals RGBA, on el quart canal indica el nivell de transparència del píxel, i s'utilitzen per exemple en aplicacions de visió nocturna.

Per norma general es capta la imatge a color, i llavors, es pot convertir a escala de grisos (també anomenades imatges d'intensitat), llavors, també es pot binaritzar. Una imatge binaria emmagatzema una matriu bidimensional on els seus bits només poden agafar dos valors, *0* (negre) i *1* (blanc). Segons l'aplicació que es vulgui fer, pot interessar més acabar obtenint una imatge final binària per treballar sobre ella, perquè l'aplicació serà més senzilla i robusta. S'aconsegueix reduir la representació de processament al mínim, i es necessita menys potència computacional, a més, permet detectar propietats geomètriques i topologies molt ràpidament. Però, per exemple, per primer detectar vores i contorns, es fa sobre una imatge d'escala de grisos.

En moltes situacions pot convenir treballar primer la imatge en color, per exemple en el cas de voler detectar un objecte o element d'un determinat color. En el nostre projecte convertim la imatge captada per la càmera a escala de grisos i a partir d'aquí la tractem per arribar a treballar amb contorns. Cal remarcar que s'ha creat una opció a l'apartat de configuració que permet treballar directament sobre la imatge en color a dos dels modes d'operació, però actualment no s'utilitza i és per estar oberts futurs experiments.

Matemàticament, una imatge fa referència a una funció d'intensitat bidimensional, i es representa com *f(x,y)*, on *x* i *y* són les coordenades espacials, i el valor de *f* és proporcional a la intensitat o nivell de gris de cada punt tractat. En el cas d'una imatge RGB serien tres funcions, una per cada canal.

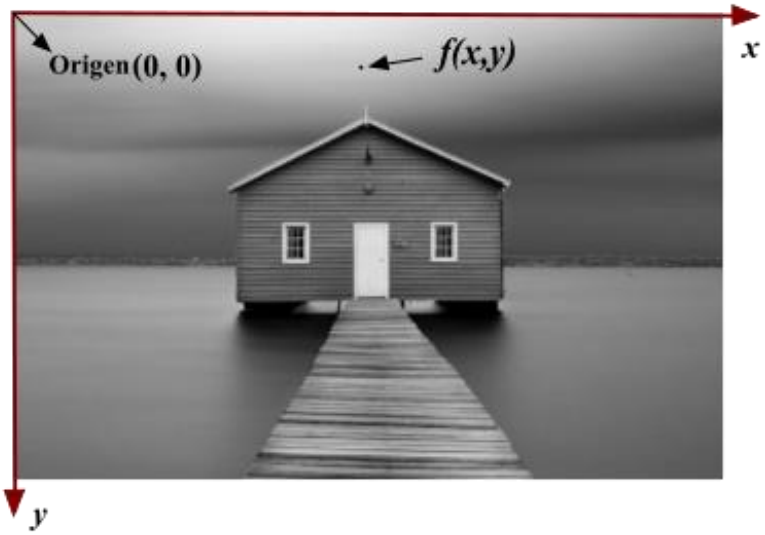

Figura 6 . Convenció d'eixos utilitzada per la representació d'imatges digitals.

Llavors, existeix el procés de segmentació, que s'explica en el següent apartat, i serveix per extreure la informació per el seva posterior extracció de característiques, reconeixement d'objectes i interpretació de l'escena. La detecció es pot fer orientada a les vores (discontinuïtat), o bé a les regions (similitud), però abans farà falta adaptar, filtrar, i transformar la imatge original mitjançant funcions de transformació o bé operadors.

2.2.4 Operacions i transformacions generals (Segmentació)

Són moltes les operacions i transformacions que es poden aplicar sobre una imatge captada, i poden treballar tant en el domini espacial com el transformat [12]. Amb el domini espacial es processen els píxels considerant la seves posicions espacials a la imatge i els seus nivells d'intensitat associats (p. ex. les operacions morfològiques, aritmètiques sobre la mateixa imatge o entre imatges, invertir, escalar, binaritzar, suavitzar, canviar tonalitats, etc.). Amb el transformat, es treballa amb els components de freqüència, i com a conseqüència necessitarà més potència de càlcul. Les transformacions en el domini freqüencial més populars, són els filtres passa alts que emfatitzen els detalls, i els passa baixos que suavitzen els detalls, i s'aconsegueixen aplicant una funció de transferència a la imatge. Per tant, primer s'han de conèixer bé les més importants per conèixer quins i perquè s'han utilitzat en el nostre treball.

2.2.4.1 Binarització

Per obtenir una imatge binària, primer es considera un llindar, i llavors píxel a píxel s'assigna *1* si el nivell de gris és igual o superior a aquest, i *0* si és inferior. Si el valor llindar es fix, serà molt difícil sempre separar bé el fons dels objectes ja que no totes les imatges estan igual de contrastades. També s'ha de tenir en compte que la variació de la il·luminació durant el transcurs del dia origina canvis als nivells de gris.

Per solucionar-ho, primer s'obté l'histograma de la imatge, i així conèixer la distribució de píxels en funció de la seva intensitat, i a partir d'aquesta informació calcular el llindar més òptim per a cada zona de la imatge. A més interessa que l'histograma sigui bimodal (les distribucions de nivell de gris clars separades dels foscos), o sigui, que no siguin plans.

L'algoritme de binarització adaptativa, per cada valor *k* de nivell de gris genera dos grups, dividint l'histograma per *k,* i calcula les seves variàncies. Després les suma, les va guardant a un vector i finalment obté el llindar de la suma mínima de les variàncies.

En aquest projecte s'utilitzen dues funcions de la llibreria OpenCV que binaritzen, i que es detallen al apartat 2.2.4.5, i en els capítols de disseny i implementació. Són, la que aplica l'algoritme de detecció de contorns *canny*, i la *threshold*, que és per fer llindarització i acabar amb un forma ben contrastada i analitzar si es tracta d'una fuita.

La funció *canny* no aplica la binarització adaptativa, però treballa amb dos llindars (màxim i mínim), que si estan ben ajustats, per aquesta aplicació obtenim els resultats desitjats. La funció *threshold* tampoc és adaptativa, ja que fins ara no ha estat necessari. Com es veurà més endavant, a la nostra aplicació s'implementen unes tècniques que no depenen molt de la separació entre els elements i el fons de la imatge, i fins ara s'han obtingut els resultats desitjats. Si en un futur tenim problemes amb la llindarització, només hem de canviar la funció *threshold* per la *adaptativeThreshold.*

2.2.4.2 Operacions aritmètiques entre imatges

Com passa amb moltes solucions en enginyeria les operacions aritmètiques més bàsiques poden ser de gran ajuda. A partir de dues imatges podem fer la resta, la suma, la multiplicació i la divisió.

Amb la resta (subtracció) [13], utilitzant un escalar com el que es pot utilitzar a la suma per aclarir, aconseguim enfosquir la imatge i desplaçar l'histograma cap a l'esquerra. Però a més, i aquí és on trobem la base per a dos dels modes de funcionament del nostre treball, es pot arribar a detectar un moviment. Si realitzem una resta entre dos imatges de la mateixa escena però en un instant diferent de temps, a la imatge resultant, es visualitzaran els elements que han canviat la seva posició, o si n'han sorgit de nous (p. ex. el cas d'una fuita). Però si l'escena canvia la il·luminació o bé la càmera es mou, s'obtindran molts elements nous que dificultaran la detecció, i ha estat un dels reptes del treball.

Figura 7. Substracció entre imatges de la mateixa escena [13].

La suma barreja les imatges i s'utilitza molt en el disseny gràfic. En el cas de sumar dues imatges reals diferents pot donar resultats amb molta saturació i

inclús en algun cas, amb intensitats fora de límit, i per aquesta raó necessita una especial cura. Però si per exemple sumem una imatge real a un escalar (o una constant), el resultat és la imatge més clara i l'histograma es desplaça cap a la dreta.

Multiplicant dues imatges no aconseguirem res molt concret, però al multiplicar una imatge per una constant augmentem la seva intensitat i l'histograma s'estira cap a la dreta. Dividint una imatge per una constant, la imatge perd intensitat, i el seu histograma s'aconsegueix. Les dues poden arribar a ser molt útils a l'hora d'adaptar imatges per posteriors processaments.

2.2.4.3 Operacions morfològiques

Les operacions morfològiques [14], també del domini espacial, s'apliquen sobre imatges binàries. Són molt utilitzades per suavitzar vores, eliminar soroll i elements no desitjats, separar o unir regions, etc. La seva base de funcionament és una matriu que s'anomena element estructurat i sol ser de *3x3*, on els seus coeficients tenen valor *0* o *1*. En funció d'aquests valors, al desplaçar l'element estructurat per la imatge, s'aplicarà erosió, dilatació, obertura i tancament. També permet aplicar els gradients morfològics que permeten realçar els canvis bruscos d'intensitat, o sigui, detectar contorns en imatges binàries.

Figura 8. Eliminació de soroll mitjançant operacions morfològiques.

2.2.4.4 Suavitzat

Amb el suavitzat es redueix soroll d'una imatge amb varis nivells d'intensitat, també s'eliminen detalls molestos i s'omplen espais buits. Es pot aplicar tant un filtre lineal con un de no lineal. Dins el domini espacial, el nivell de cada píxel és substituït per la mitjana dels valors del conjunt de píxels veïns que queden dins la màscara, però aquest filtre difumina les vores i altres detalls.
L'alternativa és un filtre de mediana no lineal. A cada píxel resultant se li assigna la mediana dels valors dels píxels veïns inclòs el seu. És molt útil per eliminar

soroll impulsiu com el de tipus “sal i pebre”. Amb aquest les vores no queden tant difuminades. Un dels més utilitzats és el filtre gaussià, que aplica una convolució píxel a píxel a la imatge original amb una màscara (o *kernel)* Gaussià.

En tots aquests els filtres, és molt important que la mida de la màscara sigui imparell per tal de tenir un píxel central, que és sobre el que s’ha de treballar.

Per aquest projecte s’ha utilitzat el filtre Gaussià per eliminar el soroll de la imatge abans d’aplicar-hi el detector de contorns. És semblant al filtre mediana però en aquest cas fa una ponderació seguint la campana de Gauss. D’aquesta manera es dona més importància als píxels més propers al central. Els que estan més lluny solen tenir menys relació.

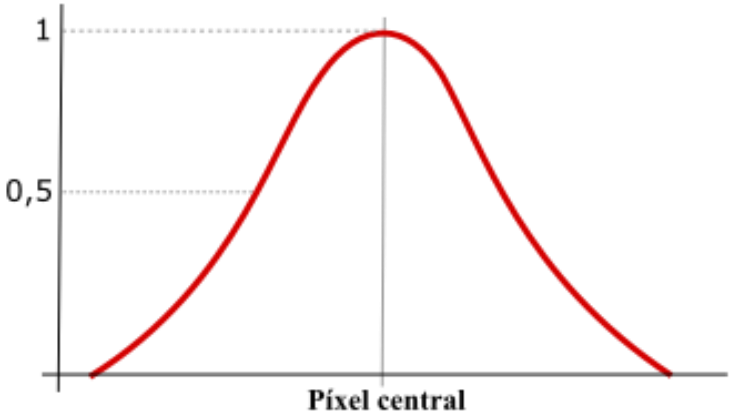

Figura 9. Funció Gaussiana aplicada a un filtre Gaussià.

La màscara de convolució d’un filtre Gaussià té el valor central més alt que la resta, i així dona més importància al píxel central que aplica la campana. Quan més gran sigui la matriu, el suavitzat serà major però la imatge tindrà menys detalls, per tant s’ha de trobar el compromís entre la reducció de soroll i la degradació de la imatge.

$$g_3 = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad g_5 = \frac{1}{246}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Figura 10. Exemple de màscares de 3x3 i 5x5 per el filtrat Gaussià.

A més té el paràmetre σ que és la desviació estàndard. A mesura que aquest valor vagi pujant, la campana s’anirà aixafant i l’efecte de suavitzat serà més gran.

$$h(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

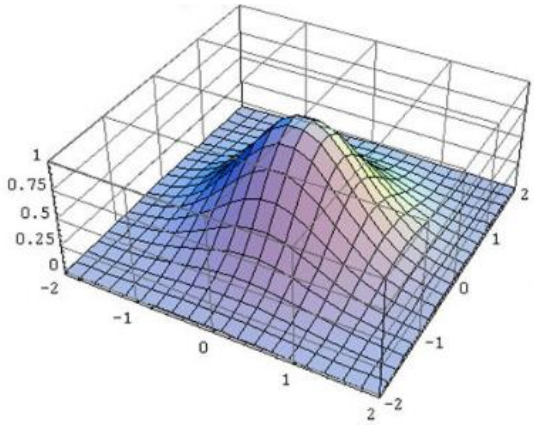

Figura 11. Expressió del suavitzat Gaussià aplicada a un píxel.

2.2.4.5 Detecció de contorns i vores

Per la detecció de contorns i vores s'utilitzen filtres que apliquen la derivada matemàtica, ja que detecten els canvis abruptes d'intensitat en totes les direccions. Els algoritmes més populars són: el *Sobel* i el *Canny* [15].

El *Sobel* és un operador diferenciador discret que calcula una aproximació al gradient de la funció d'intensitat. Es combina amb el suavitzat gaussià. A cada punt obté el vector del gradient. Es calcula per separat els canvis horitzontals i verticals, ja que hi ha un *kernel* per cada tipus, però es poden aplicar tant individualment com combinats. Les vores que retorna tenen l'amplada bastant variable.

L'algoritme *Canny*, que també es combina amb el suavitzat gaussià, aplica el gradient i fa la supressió no màxima al resultat del gradient. Després elimina els valors màxims procedents de soroll que puguin crear falses vores, i finalment tanca les vores obertes. Les vores del *Canny* són d'igual amplada, i poden ser d'un píxel.

En tots dos casos, per obtenir els contorns s'aplica un gradient que permet determinar totes les direccions del píxel que s'està tractant.

$$\overrightarrow{gradl} = \left(\frac{dl}{dx}, \frac{dl}{dx}\right) \qquad |gradl| = \sqrt{\left(\frac{dl}{dx}\right)^2 + \left(\frac{dl}{dy}\right)^2}$$

Per el nostre projecte s'ha escollit l'algoritme *canny* per vàries raons. Aquest, a diferència d'altres té un llindar intel·ligent i a per això aconsegueix un contorn molt més fi. És un dels mètodes de detecció més complexes i ben considerats.

L'Algoritme *canny* es basa en tres passos bàsics.

- Obtenció del gradient: Es calcula la primera derivada en *x* i *y*, i s'obté la magnitud i la direcció del gradient.

- Supressió no màxima: Redueix l'amplada dels contorns fins aconseguir que siguin d'un píxel. Considera quatre direccions respecte al ei horitzontal (0º, 45º, 90º i 135º).

  Per cada píxel:

  - Si el valor de la magnitud del gradient és més petit que almenys un dels seus veïns en la direcció gradient se li assigna zero.

  - En cas contrari se li assigna el valor de la magnitud.

- Histèresi del llindar: S'aplica una funció histèresi basada amb un llindar mínim i un de màxim, amb la finalitat d'eliminar l'aparició de falsos contorns.

  - Si el valor del gradient és major que el llindar màxim, el píxel s'accepta com a contorn.

  - Si és menor que el llindar mínim, no s'accepta.

  - Si està entre els dos llindars, serà acceptat com a contorn només en el cas que estigui connectat a un píxel que sigui major al llindar màxim.

2.2.4.6 Transformada de Hough

Amb la transformada de *Hough* es podran reconèixer algunes formes geomètriques com rectes, corbes, cercles, etc. Per aplicar-la, és necessari abans obtenir la imatge binària, que abans haurà passat per un filtre per detectar i realçar vores i contorn [16]. En aquest projecte, no ha estat necessari utilitzar-la, però s'ha de tenir en compte, ja que no es descarta utilitzar-la en futures versions on s'incorporin noves funcionalitats.

2.2.5 Extracció de característiques locals amb SIFT i SURF

Existeixen uns algoritmes que són molt útils en el reconeixement d'una escena, detecten i descriuen les característiques locals. Busquen la correspondència entre

diferents punts de vista. Estan pensats per aconseguir la invariància de la informació davant l'escalat, canvi d'orientació, canvi de la il·luminació, etc. SIFT (Scale Invariant Feature Transform) és més antic i necessita molt més temps de processament que el seu actual substitut, el SURF (Speeded Up Robust Features), que detecta els punts d'interès amb un cost computacional menor que el seu predecessor.

En aquest projecte no s'utilitzen, pel fet que els mètodes i tècniques que s'han escollit no ho han fet necessari. Per exemple, es treballa amb imatges fixes i per tant no s'escala ni es varia la orientació. El canvi de la il·luminació no afecta, ja que es treballa només sobre la zona susceptible a patir una fuita i no es té en compte la resta de la imatge. Si en un futur s'afegeix una funció de control remot de la càmera per tal de buscar diferents punts, llavors serà el moment de pensar en implementar un d'aquests algoritmes.

2.3 Eines i tecnologies per la implementació

Una part molt important del projecte, ha estat l'elecció de les eines i tecnologies amb que s'ha desenvolupat. Per això primer va ser necessari fer un estudi del que està oferint el mercat en la actualitat, quines són les prestacions i si existeix una bona documentació. Entre les diferents eines i tecnologies que s'havien d'elegir s'inclouen: llibreries de visió artificial, llenguatge de programació i IDE compilador.

2.3.1 Llibreries de visió artificial

Són varies les llibreries genèriques de visió per computador, unes més populars que altres, i són indispensables com a punt de partida per la implementació d'un projecte d'aquest tipus. Existeixen algunes llibreries que són obertes, i que no estan lligades a cap fabricant en concret i que es poden utilitzar amb diferents llenguatges de programació, d'altres, més tancades i que ja venen amb un programa i una interfície gràfica per la implementació de diferents aplicacions. Entre les més utilitzades es troben OpenCV, AForget.NET , Sherlock i Halcon.

2.3.1.1 OpenCV

Le sigles OpenCV [17] provenen dels termes anglosaxons *Open Source Computer Vision Library*. Aquesta llibreria gratuïta, està en una edat molt madura, ja que va néixer l'any 2000, i va ser dissenyada per Intel. És multiplataforma i té més de 500 funcions (repartides en diferents mòduls) que abasten moltes àrees de la visió artificial, sobretot el tractament d'imatges en temps real. Té l'avantatge que compte amb una gran comunitat d'usuaris, que constantment estan

desenvolupant noves aplicacions i compartint-les als fòrums especialitzats. A més, la documentació oficial és molt completa. Requereix uns bons coneixements de programació i visió artificial. Es pot utilitzar en diferents llenguatges de programació d'alt nivell com Matlab, Java, Objective C, Python, C++ i C#. Però cal destacar que en l'actualitat la majoria desenvolupadors la utilitzen amb Pyton i C++, i per tant és més fàcil obtenir suport en aquests dos entorns.

2.3.1.2 Aforget.NET

Hi ha llibreries limitades només a un únic sistema operatiu, i aquest és el cas de Aforget.NET [18], que només pot córrer sobre plataformes Microsoft i es amb programa només amb C#. Com a avantatge, té que és de fàcil ús. A més té moltes funcions per a la intel·ligència artificial incloses en un mòdul de xarxes neuronals, un de programació genètica, un d'aprenentatge automàtic i un de càlculs y operacions de conjunts difusos.

2.3.1.3 Sherlock

Sherlock [19] és un programari que es basa en una interfície gràfica que permet realitzar diferents aplicacions sense necessitat de tenir un gran coneixement en visió artificial. L'usuari ha de definir zones d'interès i definir quines funcions aplicar-hi. És de pagament i poc obert.

2.3.1.4 Halcon

Halcon [19], també de pagament, va amb el seu propi programari de desenvolupament, i també té el seu propi llenguatge. Està especialitzat per aplicacions industrials. Però a diferència del Sherlock, té unes llibreries que es poden utilitzar amb altres llenguatges de programació d'alt nivell com C, C++, C#, Visual Basic i Delphi. És molt potent, ja que té més de 1600 funcions, pot ser utilitzat des d'un usuari amb coneixements bàsics, fins a un d'expert que vulgui desenvolupar aplicacions de visió artificial molt avançades.

2.3.2 Llenguatge de programació i compilador

L'elecció del llenguatge de programació està supeditat a la de la llibreria de visió artificial, que de fet, és aquesta la base d'aquest treball. Es confiarà en un llenguatge que sigui molt utilitzat dins la comunitat de creadors, i així sigui ràpid trobar documentació i suport per la xarxa.

Després de valorar les característiques de les llibreries més populars de visió artificial, es decideix treballar amb OpenCV. Gaudeix d'una gran popularitat dins el sector, i ja porta molts anys essent el motor del molts sistemes. A més, és multiplataforma, de codi obert i gratuïta. A la pàgina oficial s'hi pot trobar tot tipus de documentació i fòrums molt actius freqüentats per usuaris molt experimentats.

La majoria de la informació que es pot trobar sobre llibreria OpenCV, està feta amb els llenguatges Python i C++. El primer, és de més fàcil ús i és ideal si es vol utilitzar en diferents sistemes operatius, però és una mica feixuc a l'hora de crear la interfícies d'usuari. En canvi, el clàssic C++ és un llenguatge molt potent i amb un gran nombre de llibreries a disposició, i també permet desenvolupar aplicacions multiplataforma. A més, és un llenguatge amb un punt de baix nivell, i permet parlar més de tu a tu amb el maquinari (si és necessàri, en assemblador). Cal destacar que els projectes més potents estan fets en C++. Per aquestes raons i perquè ja fa temps que tenia ganes de crear un projecte amb C++, ha estat l'elecció per programar aquest projecte.

Ja només faltarà elegir l'IDE i compilador. S'analitzen Builder C++ i Microsoft Visual C++. El primer és bastant complet però ha quedat clarament desbancat per la proposta de Microsoft. Per visual C++ existeix una gran quantitat d'informació. També proporciona un entorn de desenvolupament eficaç i flexible, i el seu compilador està considerat dels més eficients. Permet crear interfícies gràfiques amb diferents llibreries com ATL, MFC, Win 32 i CLR.

La solució adoptada és Visual C++ per les seves altes prestacions. Es pot descarregar de manera gratuïta el Visual Studio Community a la pàgina de Microsoft. La seva versió actual és la Visual Studio 2017.

2.3.3 Càmera de vídeo

Per crear un sistema de visió artificial hi ha diferents tipus de càmera a escollir, sempre en funció del tipus d'aplicació que es vol fer. Les càmeres lineals construeixen la imatge en continu, línia a línia. Són ideals per la inspecció en línia com per exemple una impressora industrial que imprimeix sobre paper o plàstic. Llavors, tenim les d'alta velocitat, que poden arribar a capturar milers d'imatges per segon, i aquesta seria més destinada a la investigació. Les infraroja, que ja s'han explicat prèviament, que són ideals per la visió nocturna, mapes de calor de equips electrònics, sistemes mecànics, fluids, ... però aquestes són molt cares i el nostre projecte està destinat a usuaris que busquen un sistema de baix cost. El tipus de càmera que s'adapta més al nostre projecte seria una de tipus matricial, que mitjançant sensors CCD o CMOS construeixen una matriu de píxels.

En el mercat hi ha un model que cobreix les necessitats actuals d'aquest projecte, la càmera de Logitech C920 HD Pro, amb les característiques següents [20]:

- Vídeo full HD fins 1920 x 1080.
- Compressió de vídeo H.264.
- Enfocament automàtic.
- Correcció automàtica de il·luminació.
- Connexió per USB.

Figura 12. Càmera Logitech C920 HD Pro.

3. Disseny

Amb la proposta del projecte i l'Estat de l'art, s'ha disposat d'una bona base per començar a fer el disseny del projecte. Per tant, el punt de partida del disseny han estat els objectius i l'abast marcats prèviament.

Degut a que en el camp de la visió artificial existeixen un gran ventall de tècniques, s'ha decidit fer una sistema capaç de treballar amb diferents modes (tècniques) per una mateixa finalitat. D'aquesta manera, ha permès esbrinar quina d'elles és més efectiva per assolir l'objectiu principal del sistema. A la fase de proves per obtenir els resultats experimentals, s'ha pogut comparar les diferents opcions i amb diferents productes. També, ens podem trobar que d'un determinat producte, se'n detecti millor una fuita amb una tècnica concreta. Per tant, serà interessant poder experimentar amb diferents tècniques de detecció (modes de funcionament).

Els diferents modes comparteixen una mateixa funció, que és la que crea les zones de vigilància sobre la imatge a analitzar. Aquestes zones són les susceptibles d'experimentar una possible fuita. Serien vàlvules, unions, juntes, etc. D'aquesta manera es descarten les zones i elements que no necessiten ser vigilats, i així s'evitaran falses alarmes, càlculs extres i pertorbacions en l'anàlisi.

Per facilitar la implementació, s'ha creat prèviament un diagrama de blocs que plasma de manera clara la funcionalitat de sistema. D'aquest diagrama se'n poden desgranar totes les funcions i l'estructura final del codi. A més, va acompanyat d'un disseny previ de la interfície gràfica que facilita de la millor manera possible l'ús del sistema.

3.1 Modes de detecció de fuites

S'han creat 3 tècniques per tal de detectar fuites:

- Detecció directe de contorns a la zona a analitzar.
- Subtracció entre imatge mestre fixa i imatge de temps real.
- Subtracció entre imatge anterior i imatge de temps real.

3.1.1 Detecció directe de contorns a la zona a analitzar

Aquest mode treballa sobre la imatge de temps real que proporciona la càmera. Es tracta de de convertir la imatge RGB a escala de grisos, filtrar-la per eliminar el soroll i aplicar un detector de contorns, i finalment, detectar els nous contorns que puguin aparèixer a la zona de vigilància. Aquí s'ha plantejat el repte de crear una

o dues màscares per tal de fer que només es vegin els contorns de dins les zones de vigilància. D'aquesta manera facilitem l'anàlisi, donat que no tenim informació no necessària que pugui pertorbar.

Per eliminar possibles sorolls s'ha aplicat un filtre Gaussià [21] i per la detecció de contorns l'algoritme *Canny* [22], que ja dona una imatge binaritzada. Per detectar els contorns s'utilitza una funció de OpenCV anomenada *findCountorns* [23].

A més, s'han implementat els ajustos [21] [22] que permeten modificar la desviació estàndard (σ) en el eix *x* de l'amplada de la campana del filtre Gaussià, i els llindars màxim i mínim del detector de contorns (explicat a l'apartat de l'Estat de l'Art). La mida de la màscara de convolució del filtre gaussià en principi té una mida preestablerta i fixa, la de *5x5*. Amb aquesta mida s'elimina el soroll, sense degradar la imatge.

3.1.2 Subtracció entre imatge mestre fixa i imatge de temps real

Amb aquesta tècnica es captura una imatge de la càmera, que es guarda en una en un fitxer i s'utilitza com a imatge mestre. Aquesta imatge mestre es compara periòdicament amb una imatge actual de la càmera. Si a la imatge mestre se li resta una de temps real, es pot obtenir una imatge resultant on només es mostraran els nous elements que hagin aparegut. Aquesta imatge resultant és processada amb una funció *threshold* [24] i una operació morfològica [24] per tal de poder aplicar-hi els càlculs oportuns i obtenir-ne les propietats geomètriques. D'aquesta manera es pot analitzar de manera adequada.

S'ha de tenir en compte que la imatge mestre sempre és la mateixa, i per tant, a mesura que passi el temps, degut a les variacions de la il·luminació i les ombres, poden aparèixer elements no desitjats a la imatge resultant. Per això s'ha decidit la creació de les zones de vigilància limitades al punt on es pot produir una fuita.

Hi ha un apartat on es pot captar la imatge mestre i aplicar-hi ajustos de brillantor i contrast. També un apartat on configurar les zones de vigilància. Aquestes zones de vigilància també es poden modificar quan el sistema estigui funcionant.

Per tal de tenir més possibilitats i resultats experimentals, les subtraccions tenen la possibilitat de fer-se amb les imatges a color o bé en escala de grisos. En el cas de fer-ho amb la imatge en color, també dona opció de convertir el resultat a escala de grisos, d'aquesta manera obtenim una informació més clara. També és possible desactivar la funcions de *threshold* (binarització) i morfologia. Quan més opcions, més resultats, i en conseqüència més conclusions. Es tracta d'entendre el perquè de tot.

3.1.3 Subtracció entre imatge anterior i imatge de temps real

Aquest tercer mode d'operació té la mateixa idea que l'anterior, amb la única diferència que la imatge mestre no s'ha de captar ni configurar prèviament. L'algoritme d'aquest mode, periòdicament fa una captura de la imatge mestre, i a partir d'aquí, s'apliquen les mateixes operacions que el mode de subtracció entre imatge mestre fixa i imatge de temps real.

Per tant, aquest també utilitza les zones de vigilància. També permet l'ajust ajustos de brillantor i contrast, que pot ajudar a obtenir una imatge resultant més detallada. A més també ofereix la possibilitat treballar amb la imatge directe a color i desactivar la funció de binarització i la operació morfològica, per donar més opcions a experimentar a la fase de proves.

S'ha optat per aquesta tècnica que difereix en un punt de l'anterior, perquè d'aquesta manera desapareixen les diferències no desitjades arrel dels canvis de la il·luminació i les ombres. Si anem actualitzant la imatge mestre en intervals de temps, els canvis ambientals que puguin aparèixer de manera progressiva, passaran desapercebuts.

L'interval de temps de cada quan es fa l'actualització de la imatge mestre és ajustable per part de l'usuari, d'aquesta manera es podrà trobar el valor òptim per cada situació.

3.2 Diagrama de bolcs del sistema

S'ha creat un diagrama de blocs molt detallat que desgrana totes les funcions i passos que formen el sistema. Aquest esquema ha servit de punt de partida i guia per la implementació, tant de la part de la interfície gràfica, com de la del codi.

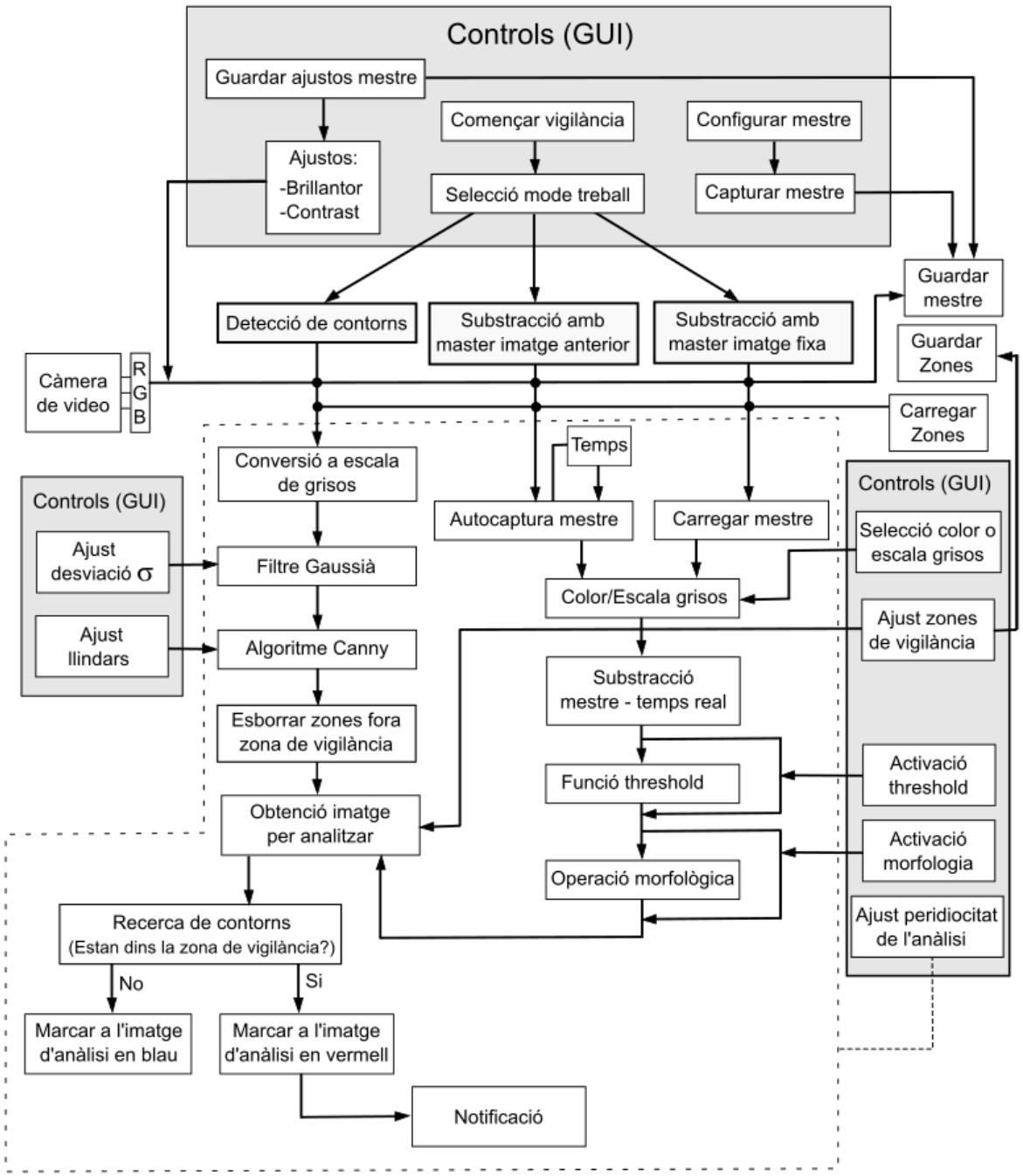

Figura 13. Diagrama de blocs del sistema.

3.2.1 Explicació general del diagrama de blocs

En el diagrama si poden diferenciar totes les parts que conformen el sistema:

- La interfície de control, que són els requadres en gris, i és la que l'usuari utilitza per gestionar tot el procés. L'usuari pot fer configuracions, ajustos, seleccionar el mode de funcionament, donar ordre de marxa del sistema, etc.

- Captura de la imatge real mitjançant una càmera que proporciona un senyal RGB. Aquesta càmera és la única font que proporciona les imatges exteriors que el sistema necessita per fer l'anàlisi, tant per capturar la imatge mestre, com per fer la captura en temps real.

- Emmagatzematge de la imatge mestre fixa i les seves dades d'ajust i que només s'utilitza per el mode de subtracció entre imatge mestre fixa i imatge de temps real. La càrrega la fa dins la seqüència d'aquest mode.

- Emmagatzematge i càrrega de les mides de la zones de vigilància (fins a dues). Utilitzat per els tres modes de funcionament, i a part, es poden configurar i assignar en temps real, o sigui, mentre el sistema estigui en funcionament.

- Execució de l'anàlisi amb el mode de funcionament seleccionat per l'usuari, que és la part que esta dins el requadre amb línia discontinua. Aquesta amb la particularitat que s'executa per interval de temps, i que també és ajustable per l'usuari en qualsevol moment.

- Obtenció de la imatge resultant on s'hi representa informació que posteriorment s'ha d'analitzar.

- Anàlisi de la imatge resultant mitjançant la recerca de contorns i les ubicacions d'aquests dins la imatge. Si es troben o no, dins la zona de vigilància.

- Notificació a l'usuari en el cas de que el sistema hagi detectat una fuita. Aquesta notificació es fa mitjançant un correu electrònic.

3.2.2 Els modes de funcionament en el diagrama de blocs

En el diagrama de blocs també si poden veure totes les parts de cada mode de funcionament, que com s'explica a l'apartat previ, s'executen periòdicament. El mode per detecció de contorns té un funcionament totalment diferent als que utilitzen la subtracció. Aquests altres modes comparteixen varis passos i funcions, i per aquesta raó s'expliquen en el mateix apartat.

3.2.2.1 Mode per detecció per contorns en el diagrama de blocs

Com s'explica de manera molt detallada a la part de l'Estat de l'Art, per aconseguir una detecció de contorns en una imatge a color, abans s'haurà de convertir a escala de grisos, seguidament suavitzar-la per treure el soroll i finalment aplicar-l'hi l'algoritme de detecció de contorns. També s'ha explicat i raonat el filtre de suavitzat i l'algoritme de detecció de contorns escollits, que són, el filtre Gaussià i l'algoritme *Canny*.

Un cop obtinguda la imatge binaritzada amb tots els contorns, se l'hi aplica la funció que elimina tots els contorns que queden fora de la zona (o zones) de vigilància. Un cop assolit aquest últim punt, ja s'obté la imatge resultant i es passa a la part d'anàlisi, que és comuna per tots els modes.

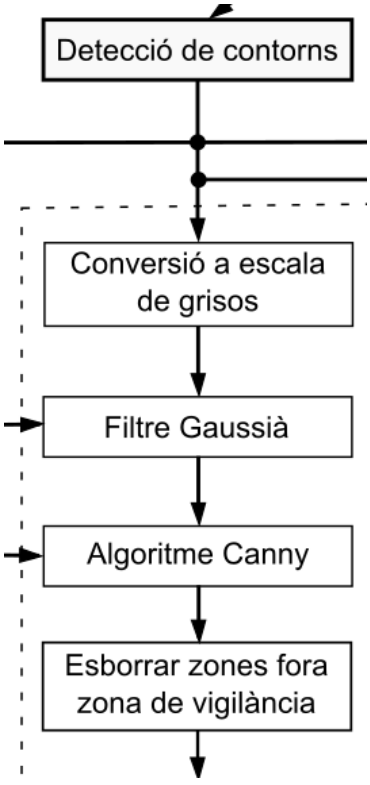

Figura 14. Mode de detecció de contorns en el diagrama de blocs.

3.2.2.2 Modes per subtracció entre mestre i temps real en el diagrama de blocs

Els altres dos modes de funcionament que utilitzen la tècnica de la subtracció (subtracció entre imatge mestre fixa i imatge de temps real i subtracció entre imatge anterior i imatge de temps real), comparteixen la majoria de passos, amb la diferència de que un sempre compara sobre la mateixa imatge mestre prèviament capturada, i l'altre la va actualitzant periòdicament de manera automàtica. Llavors, les etapes són idèntiques per els dos modes.

Primer es fa la conversió a escala de grisos, seguidament s'aplica la subtracció per obtenir una imatge amb els nous elements, llavors se li aplica la binarització i finalment una operació morfològica per tenir una figura més clara i sense gra.

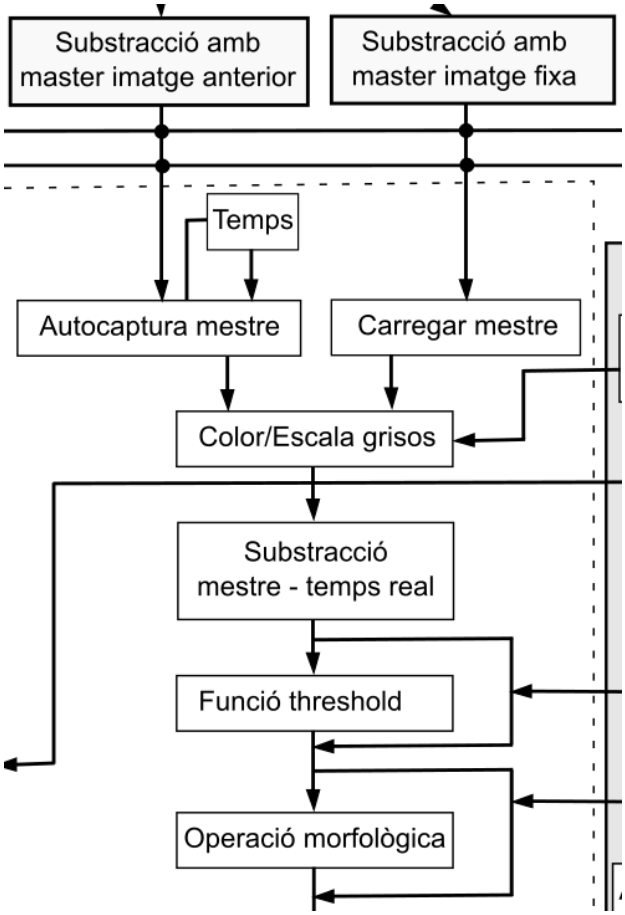

Figura 15. Modes per subtracció entre mestre i temps real en el diagrama de blocs.

Com ja s'ha comentat, es dona la opció a fer la subtracció amb la imatge a color, desactivar la funció que binaritza i la operació morfològica. Això permet experimentar i comparar, a més, entendre més la utilitat d'aquestes tècniques.

3.3 Disseny de la interfície

A partir del diagrama de blocs s'ha creat el disseny de la interfície gràfica de l'aplicació, amb la idea que sigui una interfície fàcil, intuïtiva i compleixi les necessitats bàsiques. A més, ha estat pensada per ser escalable, o sigui, que si puguin afegir noves funcionalitats en un futur sense haver de fer grans canvis.

S'ha fet un disseny sense pensar en limitacions, donat que la implementació s'ha fet amb Visual Studio C++ i CLR (Common Language Runtime) [25], que aporta a l'entorn de desenvolupament, moltes eines i controls per al disseny de formularis i panells de control.

Un cop executada l'aplicació, s'obre un panell on es poden visualitzar els controls principals, els monitors de la imatge mestre, les imatges en temps real, la resultant i l'analitzada. També, un apartat on fer tots els ajustos i configuracions.

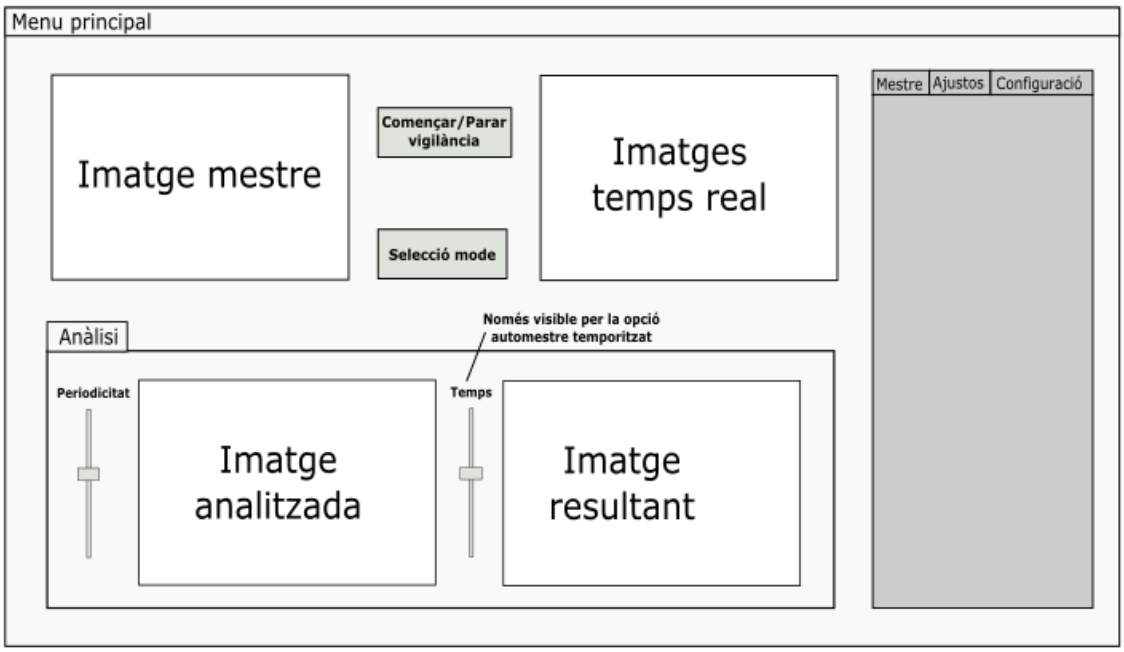

Figura 16. Disseny del panell principal.

Es reserva la part dreta del panell principal per tots els ajustos i configuracions, tant les que s'han decidit, com les futures. S'utilitza el control que permet definir varies pàgines en una mateixa àrea. D'aquesta manera quan el sistema creixi, adquirint noves funcionalitats, es podran afegir nous ajustos i noves opcions de configuració, afegint una nova pestanya o bé amb un una barra de desplaçament.

S'ha decidit separar els ajustos i configuracions en tres parts: configuració del mestre, ajustos a aplicar a les imatges i configuració de les diferents opcions.

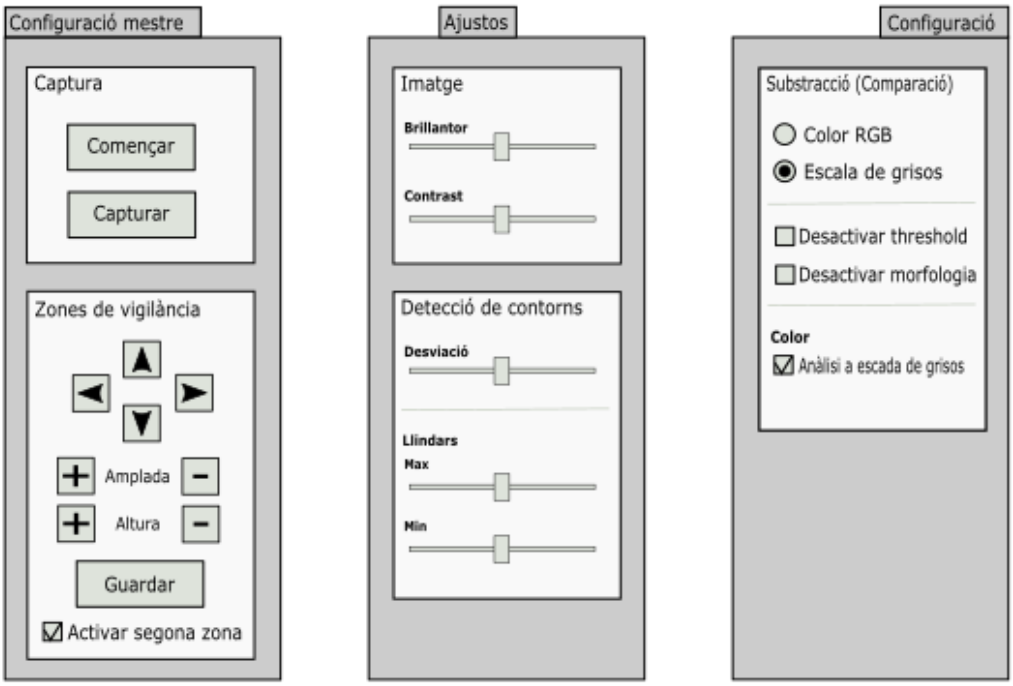

Figura 17. Disseny dels ajustos i configuracions.

4. Implementació

En la implementació s'ha traduït a codi el diagrama creat a la part de disseny, amb les eines que es van ser seleccionades en l'Estat de l'Art: el llenguatge C++ amb OpenCV i l'entorn de desenvolupament de Microsoft Visual Studio 2017. Per la interfície gràfica s'han utilitzat els controls que ofereix la plataforma .NET [26] mitjançant CLR (Common Language Runtime).

Les funcions més importants, són de la llibreria de visió artificial OpenCV, i per cada operació i transformació, s'ha buscat abans la funció més adient, conèixer com treballa, quins paràmetres utilitza, i si ha estat necessari, abans fer proves en una petita aplicació prototip a part. Un cop validada, s'ha implementat a l'aplicació final.

Per entendre millor l'estructura del programa, primer es presenta la part final de la interfície gràfica en un exemple de funcionament.

4.1 Implementació de la interfície gràfica del sistema

S'ha intentat mantenir la màxima fidelitat al diagrama de blocs representat en la figura 13, apartat de disseny, donant opcions a variacions, en funció de les proves realitzades i resultats que s'han obtingut durant el procés d'implementació.

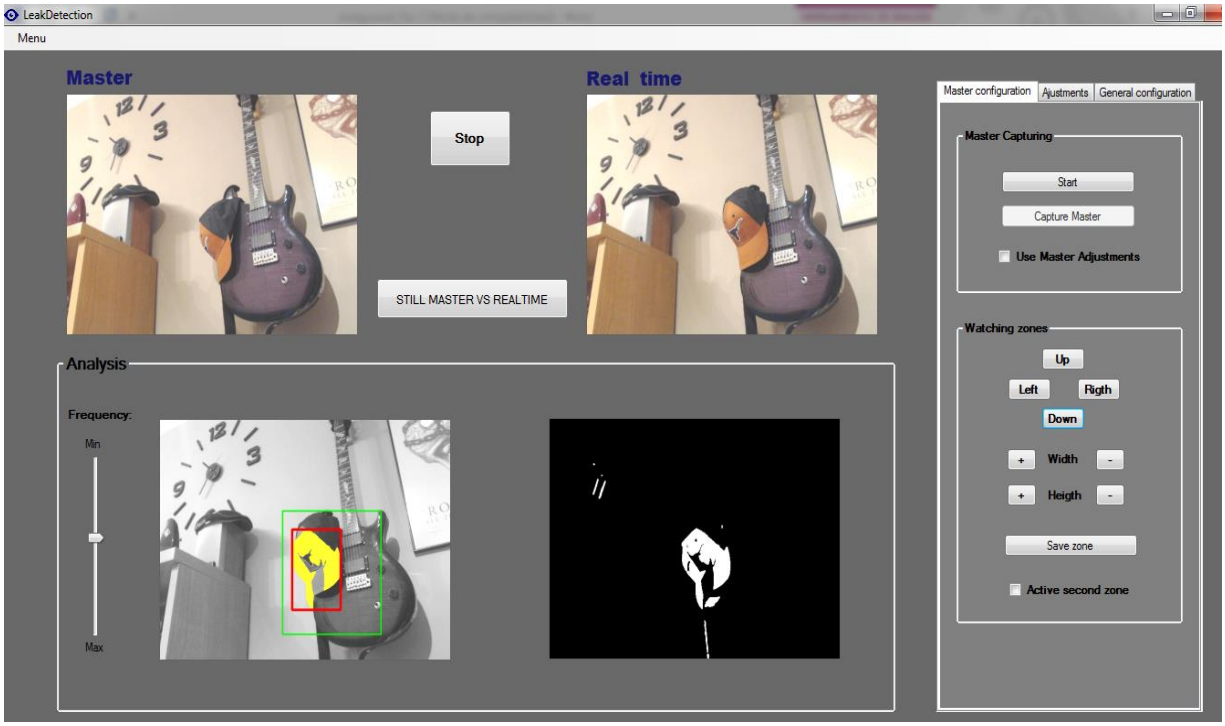

Figura 18. Captura de la interfície (treballant en mode mestre fixa).

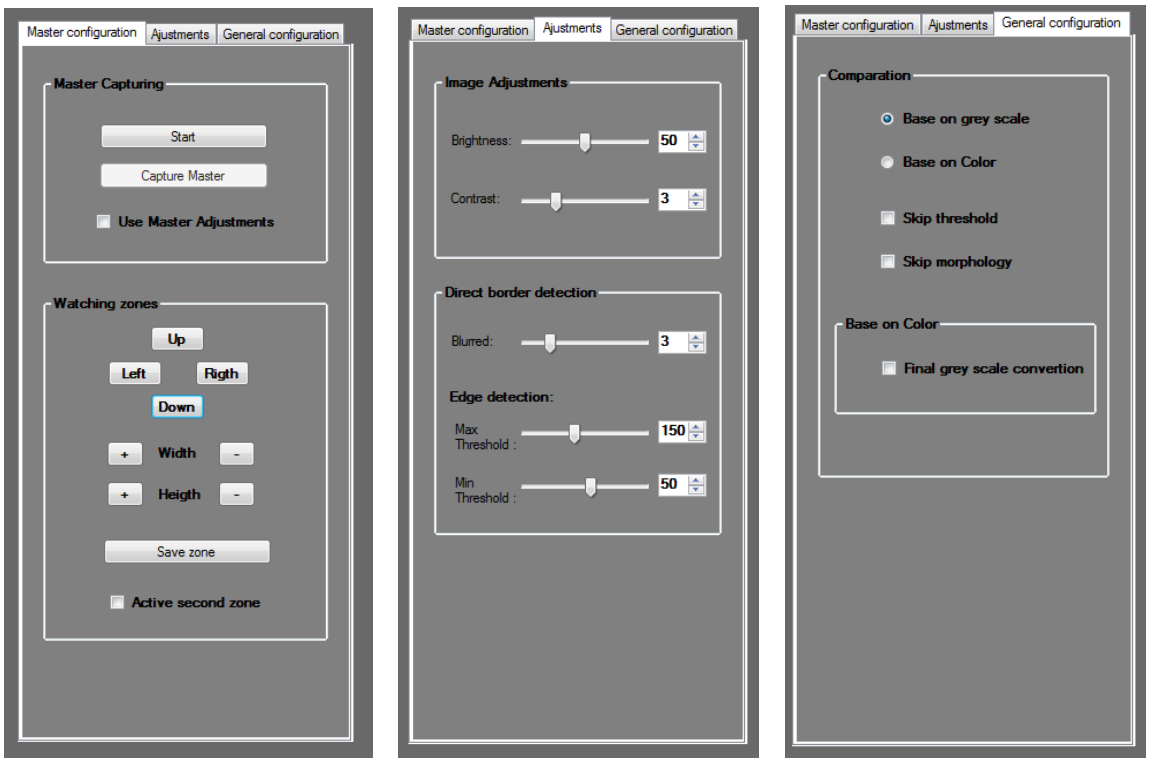

Figura 19. Ajustos i configuracions de la interfície del sistema.

4.2 Configuració de l'entorn de desenvolupament

Primer de tot, s'ha descarregat la llibreria de visió artificial a la pàgina www.open.org. La versió descarregada ha estat la v4.3.3, i un cop instal·lada ha quedat a la ubicació "*c:\openCV*". Posteriorment s'ha afegit la ubicació "*C:\OpenCV\Library\build\x64\vc15\bin"* a les variables d'entorn del Sistema Operatiu.

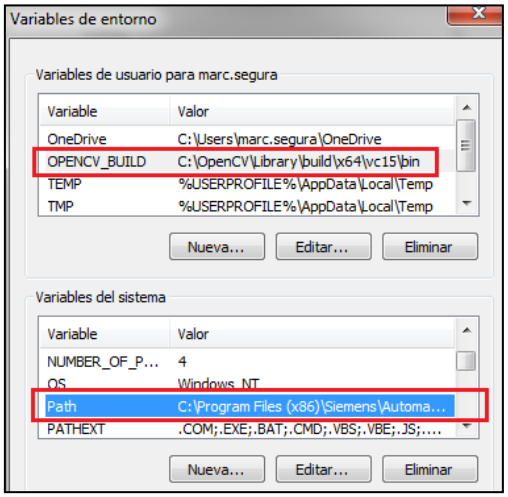

Figura 20. Configuració de la llibreria OpenCV al Sistema Operatiu.

El Visual Studio es descarrega des del lloc oficial de Microsoft, on actualment està disponible i de manera gratuïta, la versió 2017 https://visualstudio.microsoft.com/es/downloads/. Es descarrega la versió *community*.

Un cop instal·lat i creat el projecte CLR, ha fet falta vincular-lo amb la llibreria OpenCV. S'ha d'accedir a les propietats del projecte tal com mostra la figura següent.

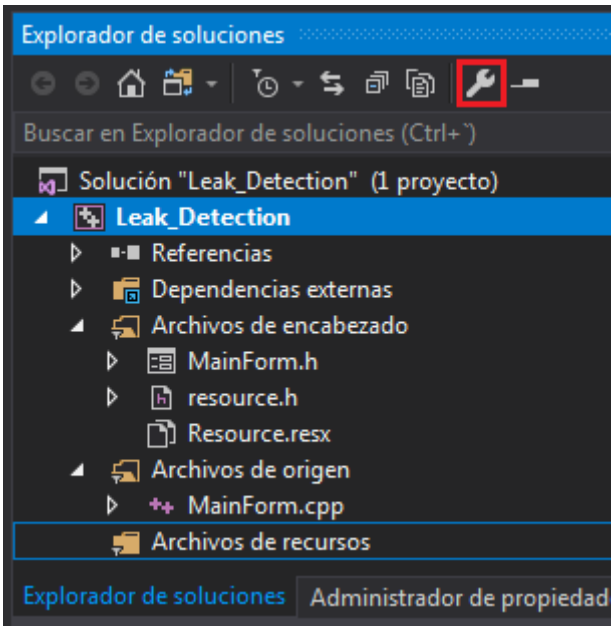

Figura 21. Accedir a les propietats del projecte.

Ens apareix una finestra on s'han de configurar tres camps amb les ubicacions de la llibreria, i a més el nom del fitxer *"opencv_world343d.lib".*Primer, a la pestanya *"C/C++ / General"* es configura el camp *"Directoris d'inclusió adicional"* amb la següent ubicació "*C:\OpenCV\Library\build\include*". S'ha de fer tant per opció *"debug"* (mode depuració) com per el "*release"* que serà l'aplicació final.

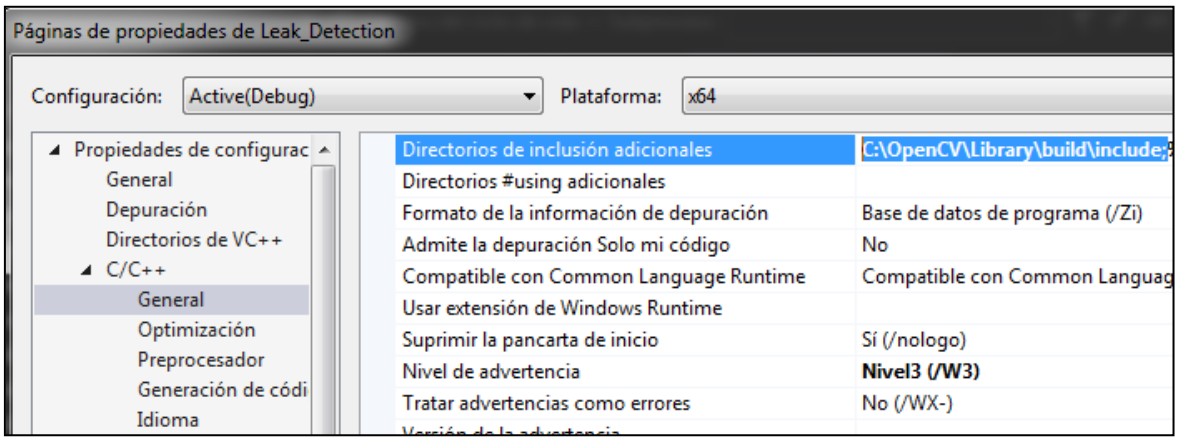

Figura 22. Configuració de OpenCV al Visual Studio 2017 C++ (part I).

Després, a la pestanya *"Vinculador / General"* es configura el camp *"Directoris de biblioteques adicionals"* amb la següent ubicació "*C:\OpenCV\Library\build\x64\vc15\lib*".

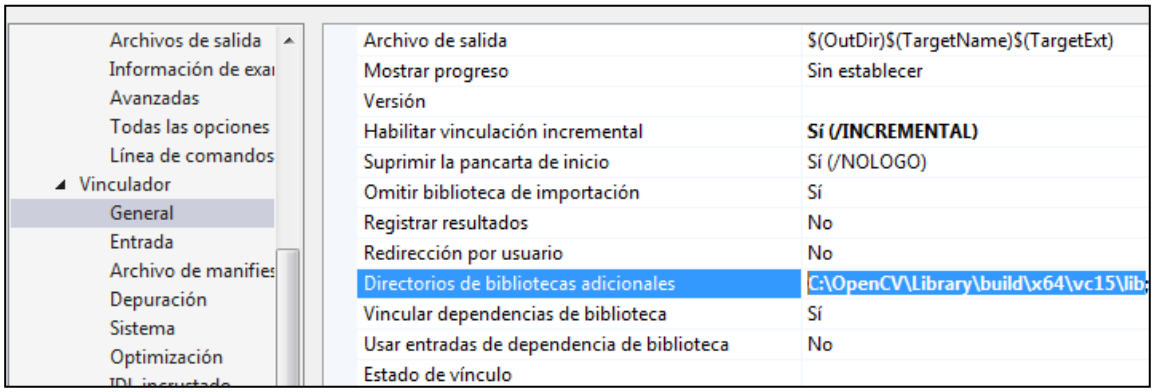

Figura 23. Configuració de OpenCV al Visual Studio 2017 C++ (part II).

I finalment, a la pestanya *"Vinculador / Entrada"* es configura el camp *"Dependencies adicionals"* amb el següent fitxer *"opencv_world343d.lib".*

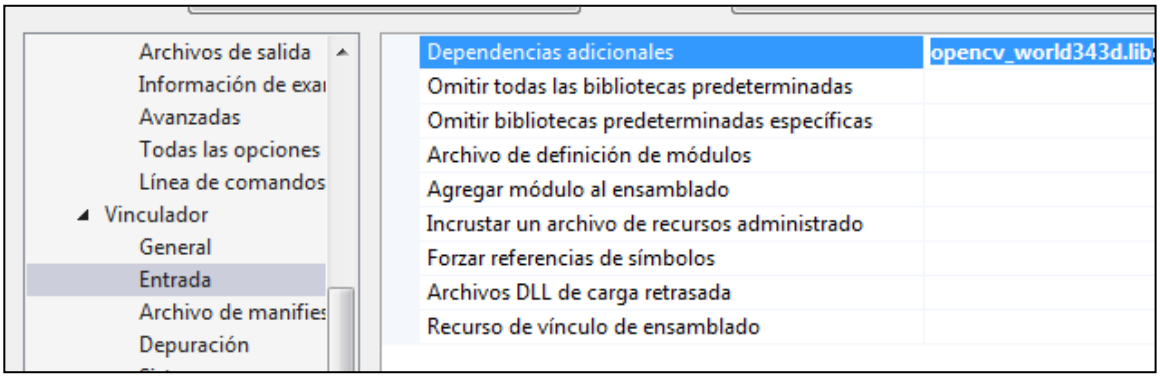

Figura 24. Configuració de OpenCV al Visual Studio 2017 C++ (part III).

En aquest punt, l'entorn de programació ja estarà lligat amb la llibreria, i es poden cridar tots els mòduls de OpenCV que es necessitin per a la implementació del projecte.

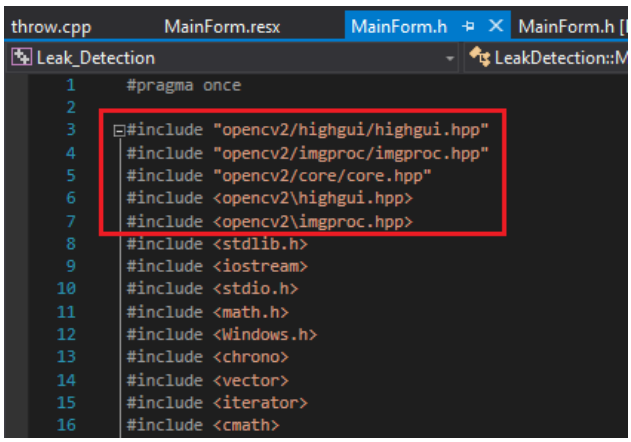

Figura 25. Inclusió de les llibreries de OpenCV a Visual Studio C++.

També s'ha afegit l'espai de noms d'aquesta llibreria juntament amb els altres, de sistema i l'estàndard. Així doncs, per utilitzar les classes i funcions de OpenCV, no fa falta posar-hi a davant "cv::". En aquest projecte s'ha decidit que "cv::" vagi precedit a tot els elements de OpenCV per poder identificar-los en el codi.

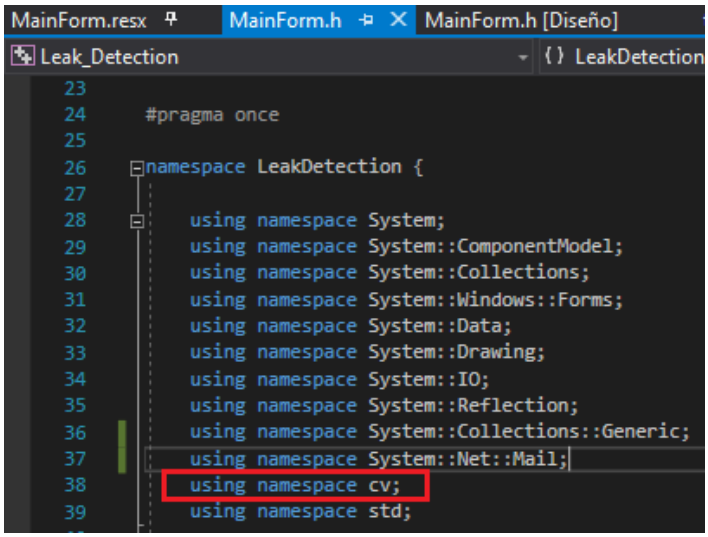

Figura 26. Vincle de l'espai de noms de OpenCV.

A més, s'han creat les carpetes *"Data"* i *"Images"* dins la carpeta *"OpenCV"*. Dins la primera, s'hi guarden els dos fitxers de text per guardar les dades dels ajustos i de les zones de vigilància. A la segona, la imatge mestre per fer la consulta quan es treballa amb el mode amb mestre fixa, i a més altres imatges del projecte com la icona de l'aplicació, la imatge dels monitors quan no estan funcionant, etc.

Dins la carpeta *"OpenCV"*, també si troba el projecte de Visual Studio 2017. Es pot provar i depurar des del projecte, però a més dins la ubicació *"C:\OpenCV\Leak_Detection\x64\Release"* si troba l'aplicació compilada i apunt per funcionar.

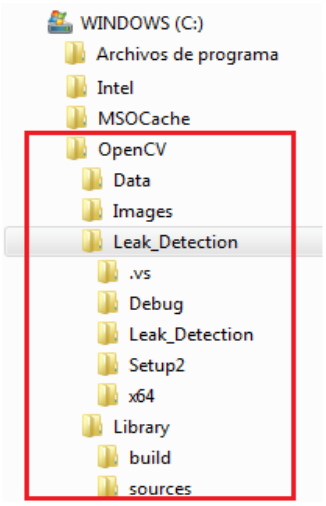

Figura 27. Ubicació del projecte dins la unitat de disc.

4.3 Estructura principal de l'aplicació

La programació s'ha fet en el fitxer de codi del formulari principal. En aquest fitxer es criden totes les llibreries que s'utilitzaran, tant les internes com les externes (OpenCV), es declaren totes les variables, s'inicialitzen tots els controls, es codifiquen els esdeveniments, és creen les funcions i les accions, etc. S'ha seguit amb la màxima fidelitat possible el diagrama de la part del disseny.

El punt de partida és l'esdeveniment que succeeix quan l'usuari prem el botó que posa en marxa i para el sistema. Primer es declaren les instàncies de les classes de OpenCV on s'emmagatzemen les dades de la captura de la càmera i la imatge que es mostrarà en el monitor (control *picture box*) de vídeo en temps real.

```
private: System::Void btnRunSystem_Click(System::Object^  sender, System::EventArgs^  e) {

        cv::Mat imgCameraRealTimeToShow;
        cv::VideoCapture capWebcamMainSystem(0);
```

En el pas següent, s'assignen els paràmetres a totes les variables i controls creats per tal de que puguin ser utilitzats en qualsevol punt de l'aplicació.

```
        if (systemRunning == false){

                if (useMasterAdjustments == true) {
                        readMasterAdjustmentsParametersFromFile();
                        }
                readWatchingZoneParametersFromFile();
```

En aquest punt, s'activa el control *timer*, i que gràcies al seu esdeveniment *tick*, anirà executant de manera periòdica l'anàlisi del mode d'operació seleccionat. A més, modificarà el text del polsador i activarà una variable per saber que el sistema està en funcionament.

```
        timer1->Enabled = true;
        this->btnRunSystem->Text = "Stop";
        systemRunning = true;
```

És aquí on s'obre la càmera, es verifica si aquesta està llesta per capturar, i en cas contrari es llença un missatge d'error amb el tipus d'incidència. Llavors comença a capturar la imatge en temps real. Per fer-ho de manera continua es crea un bucle amb la instrucció *while*, i també s'hi aplicarà l'ajust de brillantor i contrast, cridant la funció que s'ha creat al final.

```
if (capWebcamMainSystem.isOpened() == false) {
        this->lblAlarms->Text = "Error: capWebcam not accessed successfully";
        }
while (capWebcamMainSystem.isOpened()) {
boollblnFrameReadSuccessfully=capWebcamMainSystem.read(imgCameraRealTime);
```

```
if (!blnFrameReadSuccessfully || imgCameraRealTime.empty()) {
        this->lblAlarms->Text = "Error: error: frame not read from webcam";
        break;
        }
imgOriginalContrastBrightness = imageContrastBrightAdjusted(
        (Convert::ToDouble(Contrast/2)),(Brightness),imgCameraRealTime);
```

Dins aquest mateixa instrucció es fan les conversions per poder visualitzar la imatge en temps real en el control que fa de monitor. Per aconseguir-ho, s'ha de convertir del format de OpenCV a *HBitmap*, finalment a *Bitmap*. Però primer s'ha de cridar la funció que elimina els contorns que estan fora de les zones de vigilància per el mode de detecció de contorns (s'ha de fer dins el bucle principal perquè del esdeveniment *tick* no pot fer el recorregut a temps real).

```
imgCannyOnlyWatchingZone = erasePixelsOutZOneOnBinaryImg(watchingZone, watchingZone2,
imgOriginalCanny);

cv::cvtColor(imgOriginalContrastBrightness,imgCameraRealTimeToShow, COLOR_BGR2BGRA);

HBITMAP imgCameraRealTimeHBitMap = CreateBitmap(imgCameraRealTimeToShow.cols,
imgCameraRealTimeToShow.rows, 1, 32, imgCameraRealTimeToShow.data);

Bitmap^imgCameraRealTimeBitMap= Bitmap::FromHbitmap((IntPtr)imgCameraRealTimeHBitMap);
picCamera->Image = imgCameraRealTimeBitMap;
}//end while
```

I al final d'aquest esdeveniment, si es prem el polsador quan el sistema està en marxa, s'alliberen les dades de la càmera (es tanca) i es desactiva el control *timer*. A part, es modifica el text del botó i la variable del estat del sistema.

```
else // systemRunning = true;{
        btnRunSystem->Text = "Run";
        systemRunning = false;
        capWebcamMainSystem.release();
        timer1->Enabled = false;
}
```

L'execució de l'anàlisi (que és la part del diagrama de blocs de la figura 13 que està dins el requadre en línia discontinua) es fa dins l'esdeveniment *tick* del control temporitzador (*timer)*. Aquesta part s'executa periòdicament en funció del valor d'interval de temps que tingui aquest control. A cada execució, primer es declaren totes els objectes d'imatge, llavors verifica el mode de funcionament seleccionat i mitjançant la instrucció de selecció (*switch)*, s'executa el codi que correspongui.

```
private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e) {

        cv::Mat imgGrayscale;
        cv::Mat imgAnalysis;
```

```
cv::Mat imgCannyToShow;
cv::Mat imgMasterToShow;
cv::Mat imgMaster, imgResult;
cv::Mat_kernel=cv::getStructuringElement(cv::MORPH_RECT,  cv::Size2i(3,
3), cv::Point2i(1, 1));

HBITMAP imgGrayscaleToShowHBitMap;
Bitmap^ imgGrayscaleToShowBitMap;
.......
.......Altres variables
.......
HBITMAP imgAnalysisToShowHBitMap;
Bitmap^ imgAnalysisToShowBitMap;
//Master init
imgMaster = cv::imread("c:\\OpenCV\\Images\\master.jpg");
```

També, en funció de les configuració del tipus de subtracció (base a color o a escala de grisos), fa un tipus de conversió o bé un altre. A més, si el mode és el de subtracció entre imatge anterior i imatge de temps real (mode 3), crearà la imatge anterior per a la subtracció. Finalment, agafa una mostra a escala de grisos per tractar-la posteriorment.

```
try{
   //Master with delay init
   if (comparationBaseOnColor == false) {// convert to grayscale
   cv::cvtColor(imgOriginalContrastBrightness, actualFrame, CV_BGR2GRAY);
   }else {// convert to color
   cv::cvtColor(imgOriginalContrastBrightness, actualFrame, COLOR_BGR2BGRA);
   }
   if (mode3trigger == true) {
   previousFrame = cv::Mat::zeros(actualFrame.size(), actualFrame.type());
          mode3trigger = false;
   }
   //Convert to Grey scale
   cv::cvtColor(imgOriginalContrastBrightness, imgGrayscale, CV_BGR2GRAY);
```

Llavors, a la instrucció *switch,* s'executarà el codi del mode seleccionat. En aquest apartat no es posa tot el codi, ja que es mostren a l'apartat annex 1. En aquest apartat, es només pretén donar a entendre com s'estructura l'aplicació.

```
   switch (mode) {

          case 1:
                 ......
                 Codi mode per la detecció de contorns
                 .....
          case 2:
                 ......
                 Codi mode per la substracció entre imatge mestre fixa i imatge
                 de temps real.
                 ......
          case 3:
                 ......
                 Codi mode per la substracció entre imatge anterior i imatge de
                 temps real.
                 ......}
```

4.4 Configuració del mestre i les zones de vigilància

Com ja s'ha explicat a la part del disseny, per treballar amb el mode de subtracció entre imatge mestre fixa i la imatge de temps real, primer caldrà ajustar i captar la imatge mestre, a més, crear les zones de vigilància per tots els modes. Tot i així, la zona de vigilància és podrà modificar quan el sistema estigui treballant.

L'esdeveniment del polsador per començar la captura del mestre, és bastant semblant al del que posa en marxa el sistema. Crea l'objecte de la càmera, modifica el text, activa el polsador de captar el mestre, desactiva el botó de posar en marxa el sistema, modifica l'estat de la variable que diu al sistema que s'està capturant el mestre, carrega els últims paràmetres de les zones de vigilància, dibuixa aquestes zones i guarda la imatge mestre. També es controla que la càmera estigui funcionant, sinó mostra un missatge d'error a la interfície.

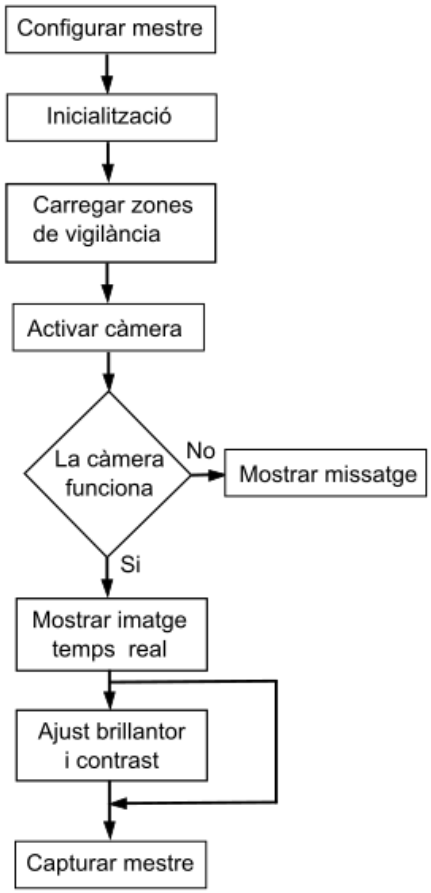

Figura 28. Diagrama de flux configuració i captura de la imatge mestre.

```
private:System::VoidbtnRunVideoMaster_Click(System::Object^  sender, System::
EventArgs^  e) {
cv::VideoCapture capWebcamForMaster(0);

if (captureMasterRunning==false) {
       this->btnRunVideoMaster->Text = "Stop";
       this->btnCaptureMaster->Enabled = true;
       this->btnRunSystem->Enabled = false;
       captureMasterRunning = true;
       readWatchingZoneParametersFromFile();
       if (capWebcamForMaster.isOpened() == false) {
       this->lblAlarms->Text = "Error: capWebcam not accessed successfully";}
```

Després es creen tots els objectes per les imatges que tracta aquesta configuració. També s'aprofita aquest apartat per visualitzar la imatge en escala de grisos i amb detecció de contorns *Canny*, i es mostren en els monitors de la part inferior. Ens seran molt útils per veure com afecten els ajustos de brillantor i contrast.

```
cv::Mat imgOriginal;
cv::Mat imgMaster;
.......
.......  Declaració de la resta d'objectes "Mat"
.......
cv::Mat imgGrayscaleToShow;
cv::Mat imgCannyToShow;
```

En aquest punt, és quan es crea un bucle per començar la captura a temps real, a color, a escala de grisos i amb la detecció de contorns. Per tant, dins aquest bucle s'utilitzaran varies funcions de conversió. Primer, es fa el control d'errors, i es crida la funció que s'ha creat per l'ajust de la brillantor i el contrast.

```
while (capWebcamForMaster.isOpened()) {
   bool blnFrameReadSuccessfully = capWebcamForMaster.read(imgOriginal);
   if (!blnFrameReadSuccessfully || imgOriginal.empty()) {
          this->lblAlarms->Text = "Error: frame not read from webcam";
          }
   imgOriginalContrastBrightness = imageContrastBrightAdjusted(
   (Convert::ToDouble(Contrast / 2)),(Brightness),imgOriginal);
```

Seguidament, s'ha codificat el condicional que gestiona quan s'ha de capturar la nova imatge mestre. Es carrega en el control *picturebox* mestre i es guarda a la carpeta *Images* del disc. L'ordre de la captura s'activa amb l'esdeveniment del botó de captura que posar a verdader la variable *captureMaster*.

```
if (captureMaster == true) {
       cvtColor(imgOriginalContrastBrightness, imgMaster, COLOR_BGR2BGRA);
       HBITMAP  imgMasterHBitmap=CreateBitmap(imgMaster.cols,  imgMaster.rows,
       1, 32, imgMaster.data);
       Bitmap^imgMasterBitmap= Bitmap::FromHbitmap((IntPtr)imgMasterHBitmap);
       picMaster->Image = imgMasterBitmap;
       imwrite("c:\\OpenCV\\Images\\master.jpg", imgMaster);
       captureMaster = false;}
```

També es fan les conversions i l'assignació que aconsegueix que es vegi la imatge en temps real amb els ajustos fets.

```
cvtColor(imgOriginalContrastBrightness, imgCaptured, COLOR_BGR2BGRA);
HBITMAP imgCapturedHBitMap = CreateBitmap(imgCaptured.cols, imgCaptured.rows,
1, 32, imgCaptured.data);
Bitmap^ imgCapturedBitMap = Bitmap::FromHbitmap((IntPtr)imgCapturedHBitMap);
picCamera->Image = imgCapturedBitMap;
```

Aquí, ja es pot fer la conversió a escala de grisos, ja que és necessari per aplicar el detector de contorns. Després, es fa el filtrat de soroll (o suavitzat) amb el filtre Gaussià, amb una màscara de *5x5* (explicat a l'apartat de l'Estat de l'Art) i la desviació estàndard que es pot variar des dels ajustos.

```
cv::cvtColor(imgOriginalContrastBrightness, imgGrayscale, CV_BGR2GRAY);

cv::GaussianBlur(imgGrayscale,imgBlurred,cv::Size(5, 5),(Blurred/2));
```

La imatge ja està preparada per la detecció de contorns per llindar per histèresis, que es centra amb dos llindars, el màxim i el mínim. D'aquesta manera es determina si un píxel forma part d'un contorn o no, donant-se tres possibles cassos: quan el valor del píxel és més gran que el llindar màxim i es considera part del contorn, quan es considera que no és contorn i el valor està per sota del llindar mínim i quan està entre el màxim i mínim i és part del contorn al estar connectat amb un píxel que forma part ja del contorn.

```
cv::Canny(imgBlurred,imgCanny,MinThreshold,MaxThreshold);
```

Llavors, es fa la conversió de la imatge suavitzada per posteriorment mostrar-la al monitor.

```
cv::cvtColor(imgBlurred, imgGrayscaleToShow, COLOR_GRAY2BGRA);
```

Per col·locar una zona de vigilància a la imatge, s'utilitza un objecte de la llibreria OpenCV, i llavors es crida la funció *rectangle* (també de la mateixa llibreria). Els paràmetres d'aquesta funció són: un objecte de la imatge, el rectangle de zona de vigilància, color, gruix, tipus de línia i canvi. En el cas de tenir activada la segona zona de vigilància, també es traça.

```
cv::rectangle(imgGrayscaleToShow, watchingZone, Scalar(0, 255, 0), 3, 8, 0);

if (activeSecondWatchingZone == true) {
	cv::rectangle(imgGrayscaleToShow, watchingZone2, Scalar(0, 255, 0), 3,
	8, 0);
}
```

I al per finalitzar el bucle, les conversions per mostrar a la interfície la imatge d'escala de grisos i la dels contorns.

```
HBITMAPimgGrayscaleToShowHBitMap=CreateBitmap(imgGrayscaleToShow.cols,
imgGrayscaleToShow.rows, 1, 32, imgGrayscaleToShow.data);
Bitmap^imgGrayscaleToShowBitMap=Bitmap::FromHbitmap((IntPtr)imgGrayscaleToSho
wHBitMap);

picRealTime->Image = imgGrayscaleToShowBitMap;

cv::cvtColor(imgCanny, imgCannyToShow, COLOR_GRAY2BGRA);

HBITMAPimgCannyToShowHBitMap=CreateBitmap(imgCannyToShow.cols,
imgCannyToShow.rows, 1, 32, imgCannyToShow.data);
Bitmap^imgCannyToShowBitMap=Bitmap::FromHbitmap((IntPtr)imgCannyToShowHBitMap
);
picCannyRealTime->Image = imgCannyToShowBitMap;

}//end while
```

Quan es torna a prémer el botó de captura del mestre, es modifica el text del botó, es desactiva el polsador de captura, s'activa el de posada en marxa del sistema i es posa la variable que avisa que s'estava en mode captura del mestre. Per acabar, s'alliberen els recursos de la càmera, o sigui que es tanca i per tant es sortirà del bucle.

```
else  {  //captureMasterRunning=true

       this->btnRunVideoMaster->Text = "Start";
       this->btnCaptureMaster->Enabled = false;
       this->btnRunSystem->Enabled = true;
       captureMasterRunning = false;
       capWebcamForMaster.release();
}
```

És molt important gravar els valors dels ajustos en el fitxer de text que es guarda dins la carpeta "*Data*", i això passa en el esdeveniment *click* del polsador per capturar el mestre, on es crida la funció que guarda els ajustos. A part de deixar la variable de mode captura activa.

```
private:System::Void btnCaptureMaster_Click(System::Object^ sender,
System::EventArgs^  e) {
if (useMasterAdjustments == true) {
       writeMasterAdjustmentsParametersToFile();
       }
captureMaster = true;
}
```

Ara només queda la part de la configuració de les zones de vigilància, que és fa mitjançant increments positius i negatius, dins els esdeveniments dels polsadors. Per el desplaçament, tenim les quatre direccions, i en funció de si tenim activada

la segona zona o no, desplaçarem un objecte o bé l'altre. Per desplaçar cap a dalt s'incrementa posició *y* a negatiu, a baix *y* positiu, esquerra *x* negatiu i dreta *x* positiu. Aquí només es mostra el codi del desplaçament cap amunt, ja que els altres tres són iguals però només canviant sentit i paràmetre.

```
private: System::Void btnWatchingZoneUp_Click(System::Object^  sender,
System::EventArgs^  e) {

	if (activeSecondWatchingZone == false) {
		watchingZone.y = watchingZone.y - 6;
	}
	else {
	watchingZone2.y = watchingZone2.y - 6;
	}
}
```

Per modificar la mida, es segueix la mateixa tècnica, modificar els atributs d'amplada i altura. Aquí es mostra el que incrementa l'amplada.

```
private: System::Void btnWatchingZoneWidthInc_Click(System::Object^  sender,
System::EventArgs^  e) {
	if (activeSecondWatchingZone == false) {
	watchingZone.width = watchingZone.width + 6;
	}
	else {
	watchingZone2.width = watchingZone2.width + 6;
	}
}
```

Finalment, a l'esdeveniment del botó per gravar les dades de configuració de les zones de vigilància, es crida la funció creada per aquesta tasca.

```
private: System::Void btnSaveWatchingZone_Click(System::Object^  sender,
System::EventArgs^  e) {
	writeWatchingZoneParametersFromFile();
}
```

4.5 Implementació de les funcions

Al final del codi del formulari s'han creat les funcions que són susceptibles a repetir-se a diferents parts del projecte. També ajuden molt a tenir un codi més net, clar i ordenat. En aquest apartat només es mostra de seva declaració, els paràmetres que utilitza i les dades que retorna. Les funcions amb el codi complet es posen a l'annex 2.

4.5.1 Ajust de la brillantor i contrast de la imatge

Per l'ajust de brillantor i contrast recorrerem tots els píxels de la imatge (guardada com a matriu bidimensional), mitjançant tres instruccions *for* recorreguts (un per cada columna i l'altre per cada fila), i un per els tres canals RGB. Els dos ajustos es faran amb la mateixa expressió, aplicada al píxels de l'entrada en cada pas.

L'expressió té dues constants, una que multiplica (α), i l'altre que suma (β), la primera afecta en el contrast, i la segona la brillantor. La que multiplica ha de ser més gran de zero i estar per sota de tres. La que suma entre el rang (0-100).

$$G(y,x) = \alpha \cdot f(y,x) + \beta$$

```
cv::Mat imageContrastBrightAdjusted (double alpha, int beta, cv::Mat image){
        ....
        Codi de la funció
        ....
return new_image;}
```

4.5.2 Escriure els valors de les zones de vigilància al fitxer de text

Gràcies a les classes de la plataforma *.Net* que permeten tractar fitxers plans, es crea el fitxer on es guarden les dades de les zones de vigilància (ubicacions i mides).

```
void writeWatchingZoneParametersFromFile() {
        ....
        Codi de la funció
        ....}
```

4.5.3 Llegir els valors de les zones de vigilància

Aquesta funció recupera la informació del fitxer de text que s'ha actualitzat des de l'esdeveniment del polsador per guardar aquestes dades mitjançant la funció d'escriptura.

```
void readWatchingZoneParametersFromFile() {
        ....
        Codi de la funció
        ....}
```

4.5.4 Escriure els valors dels ajustos de la imatge mestre

Amb aquesta funció, i tal com es fa amb les dades de les zones de vigilància, es guarden els valors dels ajustos de la imatge mestre. També s'hi ha afegit els que afecten a la conversió a contorns (desviació i llindars).

```
void writeMasterAdjustmentsParametersToFile() {
        ....
        Codi de la funció
        ....
}
```

4.5.5 Llegir i assignar els valors dels ajustos de la imatge mestre

A més de recuperar els valors dels ajustos, aquests són assignats als controls.

```
void readMasterAdjustmentsParametersFromFile() {
        ....
        Codi de la funció
        ....
}
```

4.5.6 Eliminar els contorns fora de les zones de vigilància

Aquesta funció és clau per el mode de detecció de contorns, perquè elimina tots els contorns de fora les zones de vigilància per tal que no hi hagin pertorbacions ni falses deteccions. El que passi a les zones de la imatge que no s'estan analitzant, s'ha d'eliminar. Aquesta funció ho aconsegueix posant els píxels de fora les zones de vigilància a zero (negre), i mantenint de la imatge original els contorns que es troben dins de la zona de vigilància.

La funció rep la imatge original que prèviament se li ha aplicat l'algoritme *Canny*, i les dues zones de vigilància. Dins la funció es crea una nova imatge totalment negre de les mateixes mides que la de l'entrada, i que agafarà només els píxels de dins les zones de vigilància. De la segona zona, només en el cas que estigui activada. La funció retorna aquesta nova imatge.

```
    cv::Mat erasePixelsOutZOneOnBinaryImg(cv::Rect watchingZone, cv::Rect
    watchingZone2, cv::Mat image)
       {
       ....
       Codi de la funció
       ....
       return new_image;
}
```

4.5.7 Creació del rectangle de la zona del contorn detectat

Es tracta d'una funció que s'utilitzen el els tres modes de funcionament. A partir de la imatge resultant, mitjançant la funció *findContours* de OpenCV. Aquesta funció està parametritzada per tal que s'obtinguin tots els contorns i els organitzi per jerarquia: en el nivell superior es troben els de límits externs, i a l'altre nivell els límits dels forats (paràmetre CV_RETR_CCOMP). Amb l'últim paràmetre de la funció, s'eliminen tots els punts redundants (paràmetre CV_CHAIN_APPROX _SIMPLE).

Finalment, s'agafa el contorn amb l'àrea més gran, que no podrà ser més gran que les àrees de les zones de vigilància, i es crea el rectangle que senyalitzarà el contorn detectat. A més, s'ha declarat una variable d'entrada amb el número de mode d'operació per si més endavant és necessari crear alguna tasca especial per algun dels modes.

```
void boundingRectCreation(cv::Mat imgToCheck, std::vector < std::vector < 
cv::Point > > &contours,int &largest_contour_index, Rect &bounding_rect, 
intmaximumCntournArea , bool &resetContourns, int mode){
	....
	Codi de la funció
	....
}
```

4.5.8 Notificació d'alarma de fuita mitjançant correu electrònic

Quan es detecta un nou contorn dins d'una zona de vigilància, es crida una funció que està programada perquè un cop hagi acumulat un número determinat d'avisos (per evitar falses alarmes), enviarà un avís via correu electrònic a l'usuari.

Amb les classes *Mailmessage* i *SmptClient* es configura i s'envia el missatge d'alarma a l'usuari. A més, s'activa la visibilitat del polsador de reconeixement de la alarma.

```
void notification(){
	....
	Codi de la funció
	....
}
```

5. Proves experimentals i resultats

Les primeres proves de l'aplicació, s'han desenvolupat amb unes proves experimentals, que han intentat simular un entorn industrial. Primer s'ha col·locat un tub en horitzontal i s'han provocat unes fuites amb oli lubricant. Després s'han fet les proves amb aigua, provocant una fuita en una vàlvula d'un dipòsit. Les proves s'han fet per els diferents modes de funcionament.

Al ser la primera prova real, només s'aplica una zona de vigilància, ja que en una primera fase, la finalitat és la de de comprovar si es compleixen els objectius en la detecció de fuites de fluids. Aquestes primeres proves es fan amb base a escala grisos i amb la imatge resultant binària i operada morfològicament. Si amb algun producte hi ha dificultat per a detectar, llavors es farà amb base a color i amb la imatge resultant a escala de grisos. Fins ara no ha fet falta.

5.1 Proves amb oli lubricant per detecció de contorns

Per una simulació més real s'han col·lat diferents objectes i també s'ha provocat un raig lluminós. D'aquesta manera es valida que la funció que elimina els contorns que estan fora de la zona de vigilància funcioni correctament, i per tant només s'analitza la zona susceptible a patir una fuita.

A la captura següent, es pot comprovar la gran quantitat de contorns que es poden trobar en una imatge. Aquesta, és una captura feta amb l'ajust del mestre, que també ens serveix per ajustar la detecció de contorns.

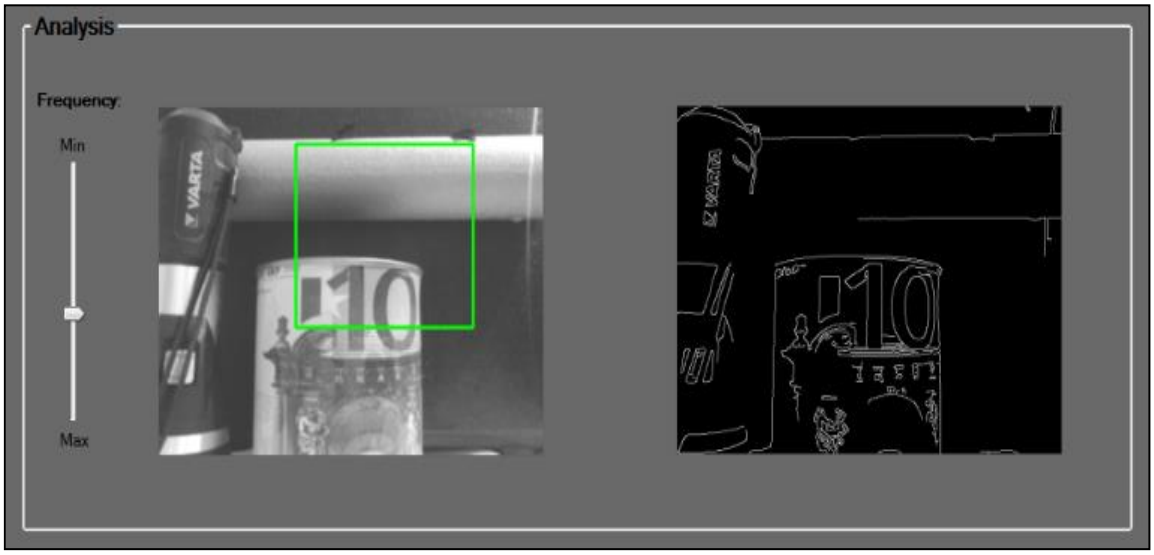

Figura 29. Detecció de contorns d'una imatge a l'aplicació.

Un cop el sistema es posa en marxa amb el mode de detecció de contorns, i amb la zona de vigilància ubicada a la zona on es pot produir la fuita, ja no hi hauran pertorbacions que puguin perjudicar l'anàlisi.

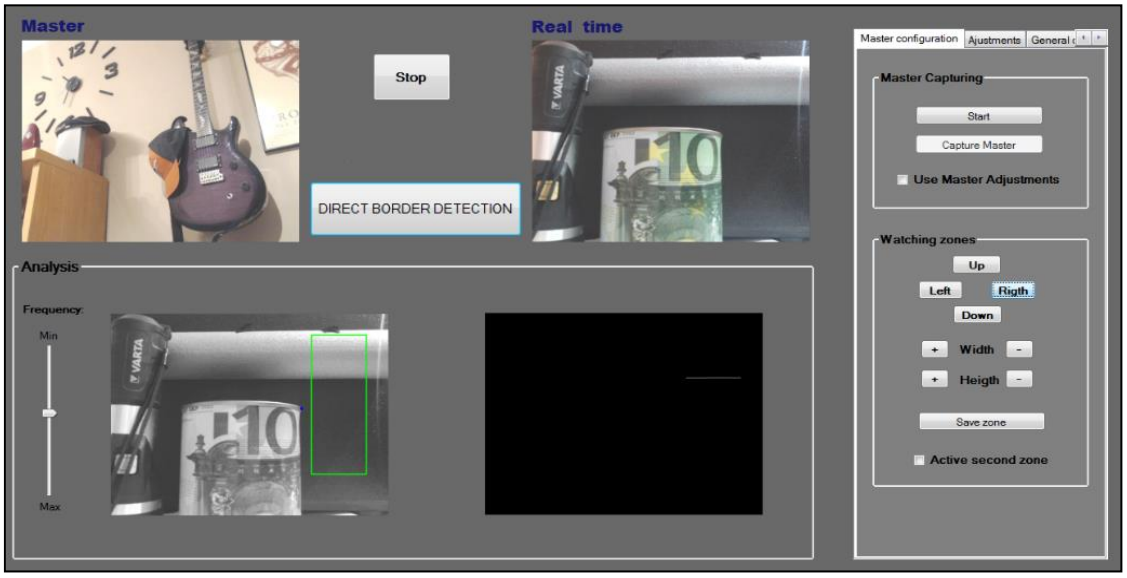

Figura 30. Ajust zona de vigilància en mode detecció directe de contorns.

Quan ha succeït una fuita, aquesta ha estat detectada al mateix moment, i l'algoritme que sobreposa el requadre vermell a la zona de la fuita, també ha dibuixat (en groc) dins el requadre, el contorn d'aquesta. També s'ha rebut la notificació per correu electrònic (actualment la notificació es fa a les 20 lectures de fuita).

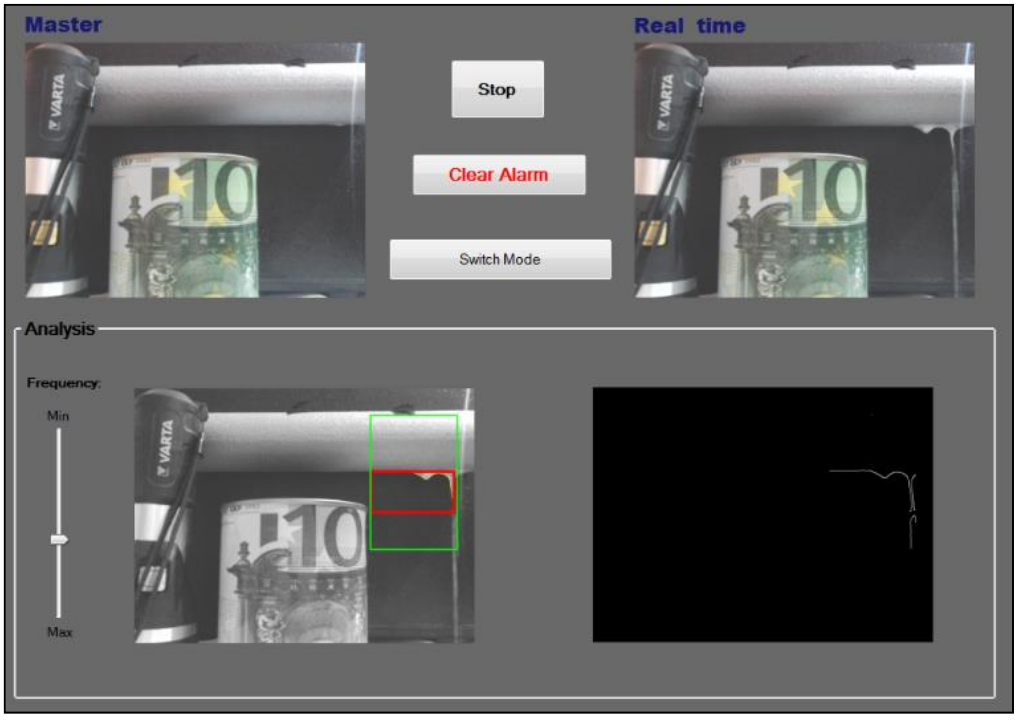

Figura 31. Detecció d'una fuita d'oli lubricant en mode detecció directe de contorns.

5.2 Prova amb oli lubricant per subtracció entre la imatge mestre fixa i la real

Abans de posar en marxa el sistema amb aquesta tècnica, primer s'ha capturat la imatge mestre fixa que es compararà en tot moment amb la de temps real. S'ha ajustat la brillantor i contrast per tal de tenir la imatge el més nítida possible i que la il·luminació no hi tingui molt d'efecte (per exemple, que no hi hagin zones cremades).

La fuita també s'ha detectat en el mateix instant de la seva aparició, quedant molt més definida que en el cas anterior. També, s'ha observat que a mesura que passava el temps i la il·luminació es veia modificada, han aparegut components no desitjats (en forma de gra), a la imatge a analitzar, com si de soroll es tractés.

Si aquests elements no passen de ser petits grans, no afectaran en l'anàlisi, inclús és podria aplicar una operació morfològica per eliminar-los. Però si la il·luminació varia de manera més agressiva, aquests components no desitjats s'aniran accentuant, poden arribar a ser més grans que la mateixa fuita, i actuar com a veritables pertorbacions, i obligar a fer una nova captura de la imatge mestre.

Figura 32. Detecció d'una fuita d'oli lubricant en mode subtracció entre la imatge mestre fixa i la real.

5.3 Prova amb oli lubricant per subtracció entre la imatge anterior i la real

La base de funcionament d'aquest mode és molt semblant al del cas anterior. Com s'ha explicat a la part de disseny, només difereix en que aquí no fa falta fer la captura manual prèvia del mestre, ja que aquesta es farà de manera automàtica periòdicament (ajustable de 5 a 10 cicles d'execució del *timer*). Per tant, aquest mode s'hauria de veure com una millora respecte l'anterior.

El fet que el mestre es vagi actualitzant, fa que si la imatge en temps real es veu modificada per l'efecte de la il·luminació o altres circumstàncies com el pas de persones, animals, efectes mediambientals, moviments d'objectes, petit moviment de la càmera, etc. l'anàlisi no es veuria afectat.

En la primera prova, la fuita també s'ha detectat en el mateix instant de la seva aparició, igual que en el cas del mestre fixa, ja que aquesta part funciona exactament igual.

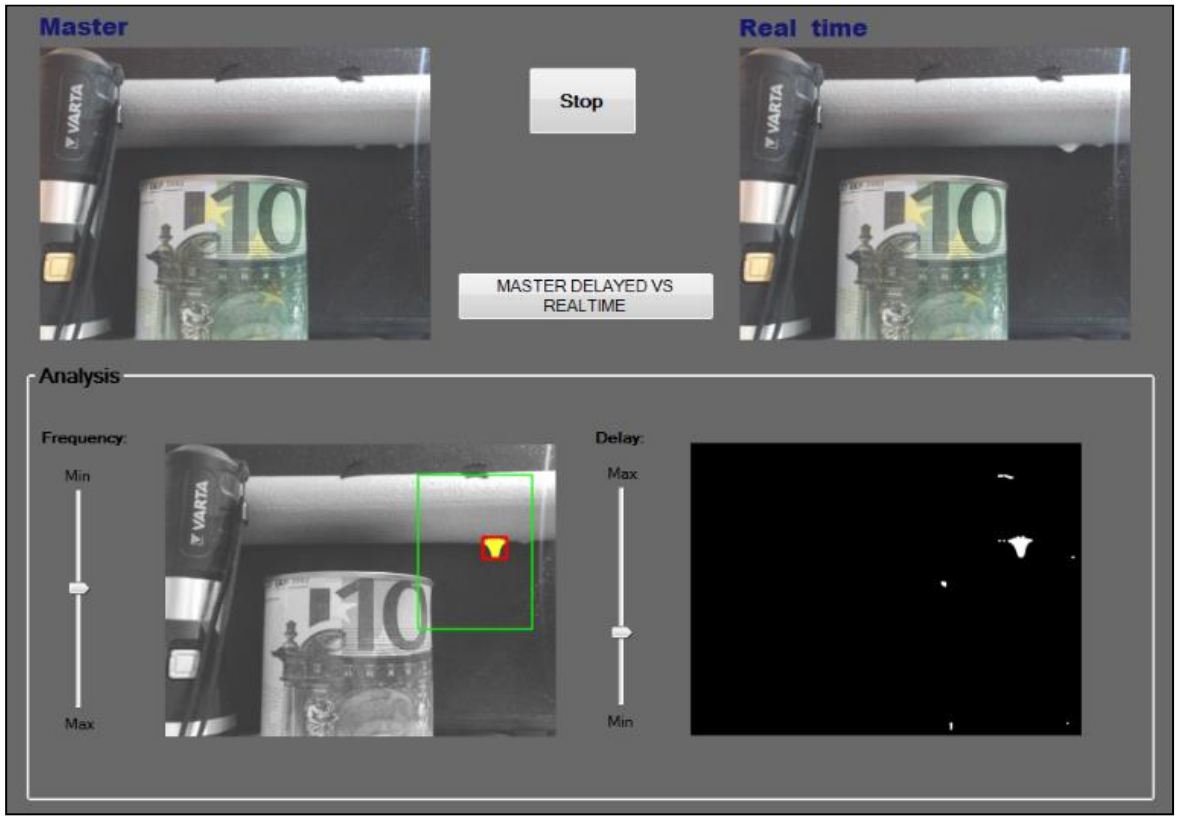

Figura 33. Detecció d'una fuita d'oli lubricant en mode subtracció entre imatge mestre anterior i la real.

A més, s'ha provocat una modificació en l'escena, per assegurar que l'anàlisi no es veurà pertorbat. S'ha extret un dels objectes, i quan ha passat el període d'actualització del mestre, aquest s'ha capturat de nou. En aquest moment, el sistema s'ha estabilitzat, i el sistema ha continuat treballant amb tota normalitat.

Quan ha succeït una altre fuita, s'ha detectat a l'instant, tal com es pot observar a la captura següent.

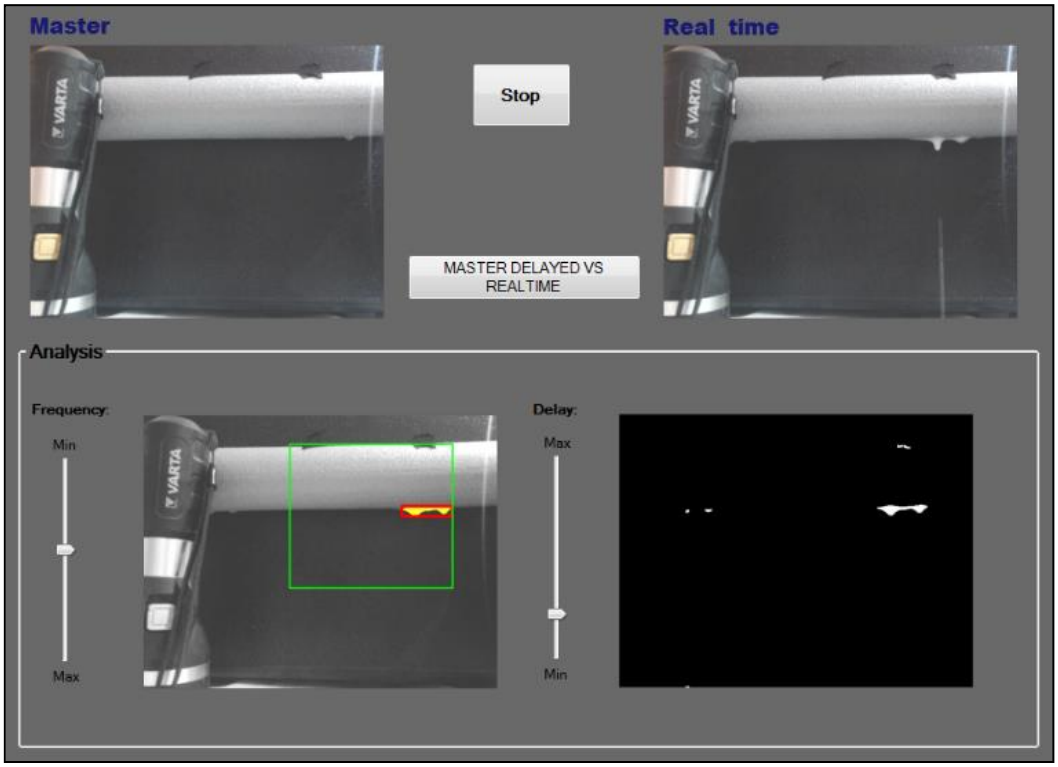

Figura 34. Detecció d'una fuita d'oli lubricant en mode subtracció entre la imatge mestre anterior i la real modificat.

5.4 Proves amb aigua per detecció de contorns

S'ha provocat una fuita per degoteig d'aigua en una vàlvula manual d'un dipòsit d'inoxidable i per el mode per detecció de contorns, el sistema l'ha detectat.

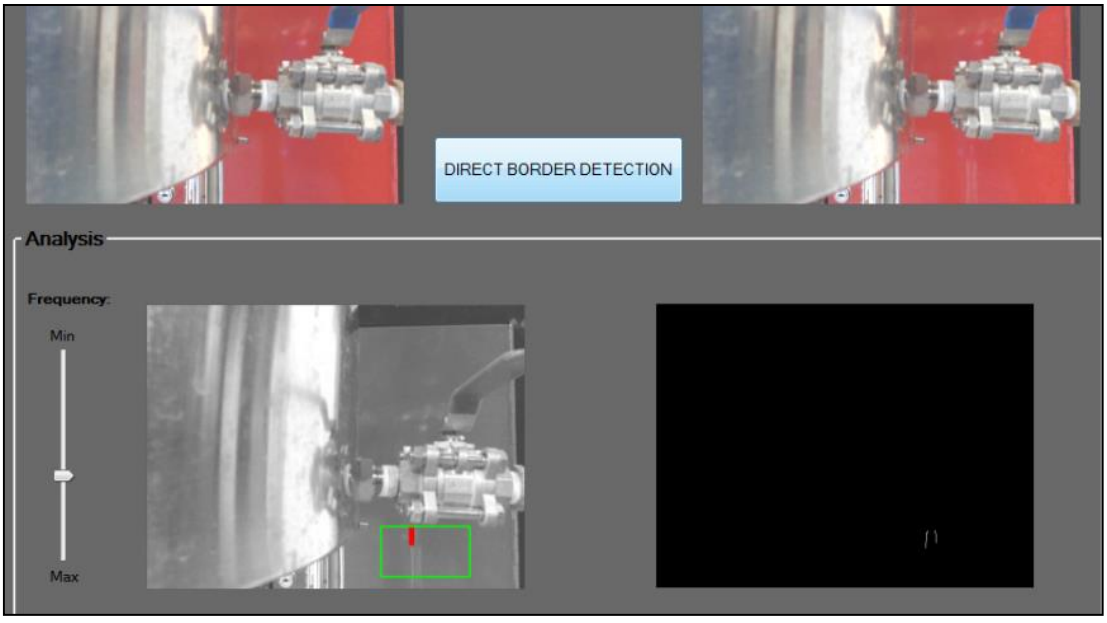

Figura 35. Detecció d'una fuita d'aigua en mode detecció directe de contorns.

La detecció no ha estat tant instantània com en el cas de l'oli, però al sistema no li ha fet falta molts cicles d'anàlisi per donar l'alarma i enviar la notificació. A més, per tenir èxit, ha sigut molt important la ubicació de la zona de vigilància, ja que si s'agafa part de la vàlvula, el sistema agafarà els seus contorns com una fuita.
També, al tractar-se d'un degoteig, s'ha comprovat que si es baixa l'interval d'execució (l'ajust *frequency* de l'esquerra) del control *timer*, es capta més ràpid la fuita. Si el sistema analitza amb més lentitud, pot captar imatges on no hi ha degoteig i tardarà molt més a donar la notificació.

5.5 Prova amb aigua per subtracció entre la imatge mestre fixa i la real

Amb el mode que es compara amb la imatge mestre fixa també hem obtingut la notificació de la fuita d'aigua.

Com en el cas de l'oli, també apareixen components no desitjats, degut a les modificacions sorgides en l'ambient passat un temps d'haver capturat la imatge mestre.

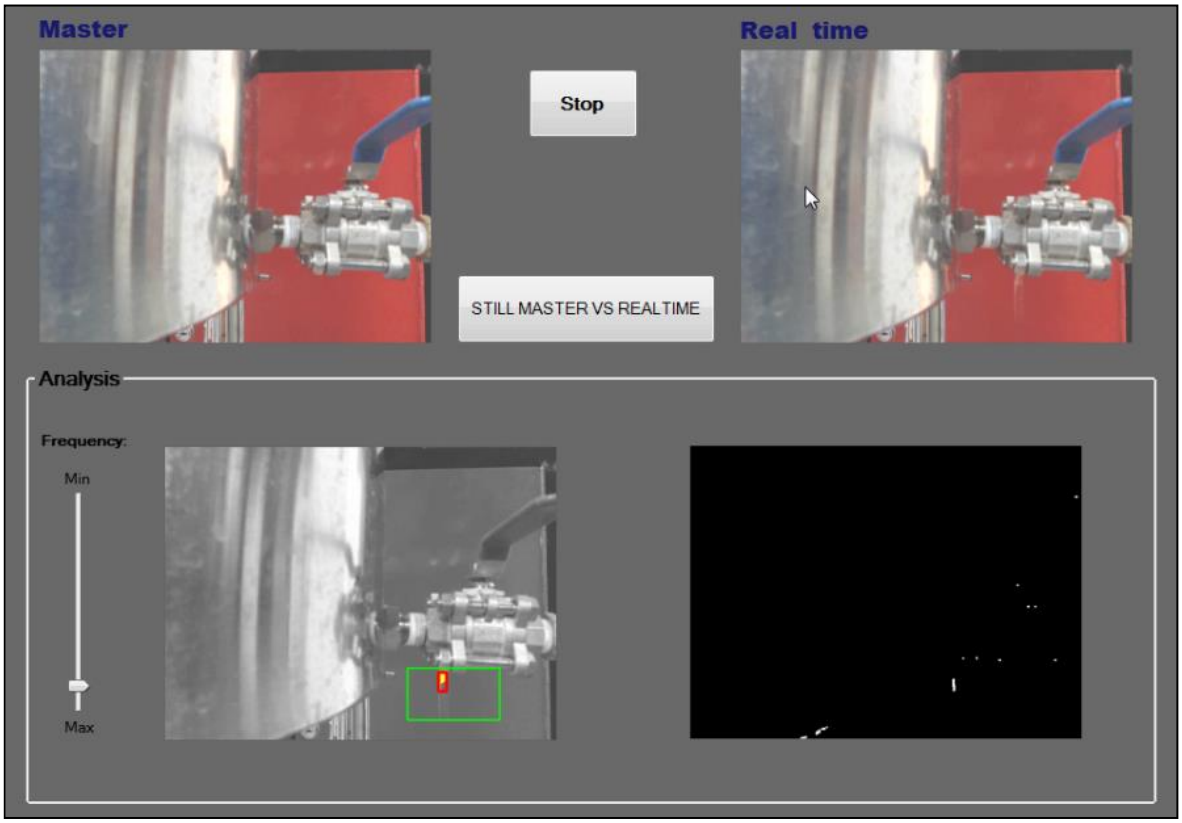

Figura 36. Detecció d'una fuita d'aigua en mode subtracció entre la imatge mestre fixa i la real.

5.6 Prova amb aigua per subtracció entre la imatge anterior i la real

Igual que en el cas de l'oli lubricant, amb el mode que actualitza de manera automàtica la imatge mestre, tenim un comportament molt net i robust del sistema i la fuita es detecta amb èxit.

No tenir components no desitjats, fa que el sistema només hagi d'analitzar els contorns de la fuita real. A més, amb aquest mode no fa falta crear una zona de vigilància molt ben ajustada a la zona a analitzar.

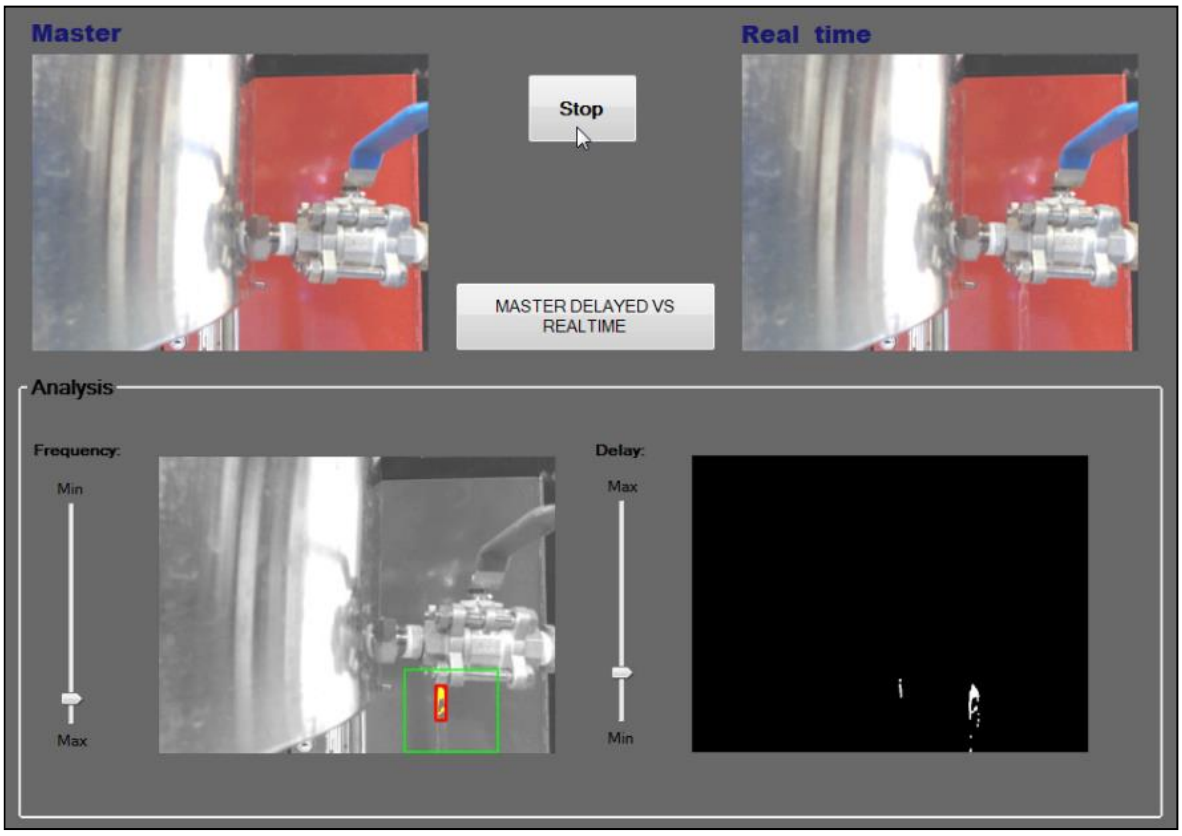

Figura 37. Detecció d'una fuita d'aigua en mode subtracció entre la imatge mestre anterior i la real modificat.

6. Conclusions i línies de treball futur

Un cop acabat aquest treball, és hora d'extreure'n les conclusions en funció de l'assoliment dels objectius marcats inicialment. També, com que es tracta d'un projecte viu, i en realitat ens trobem a una primera fase d'un projecte d'aquestes característiques, s'ha de pensar en les línies de futur, per tal de millorar l'aplicació i fer que algun dia pugui arribar a ser un producte professional i real dins el mercat.

6.1 Conclusions

En funció dels resultats obtinguts en les proves experimentals, s'extreuen les següents conclusions:

- Els tres modes de funcionament del sistema han superat les proves experimentals, detectant des de productes més opacs fins a productes transparents, com és el cas de l'aigua. Però el mètode més efectiu ha estat el de subtracció entre imatge anterior i imatge de temps real.

- En el mode de detecció directe de contorns a la zona a analitzar, si han trobat algunes limitacions. La zona de vigilància ha d'estar completament neta, ja que si agafa elements que generin contorns, llavors falseja la detecció de la fuita. A més, quan es detecta una fuita, el contorn no queda tant definit com en les altres tècniques.

- El mode de detecció de subtracció entre imatge mestre fixa i imatge de temps real és massa susceptible a les modificacions ambientals, sobretot, la causada per el canvi en la il·luminació, que passat un temps considerable de la captura de la imatge mestre, apareixen masses elements no desitjats. Això obliga a que la zona de vigilància estigui molt ben ajustada.

- També, amb el mode de imatge mestre fixa, si la càmera es mou, encara que sigui tant sols un mil·límetre, la imatge resultant de la subtracció mostrarà molts elements indefinits i abstractes, impossibilitant la detecció i obligant a l'usuari capturar la imatge mestre de nou.

- El mode que ofereix millors resultats (subtracció entre imatge anterior i imatge de temps real), s'adapta a qualsevol canvi ambiental que pugui aparèixer en el transcurs del temps, degut a que fa una autocaptura periòdica de la imatge mestre, i per tant, obvia els canvis que van apareixent a la imatge.

- Si ens trobem davant un producte poc viscós com és el cas de l'aigua, amb probabilitats de que la fuita serà un degoteig, la fuita es detectarà més ràpid si es puja la freqüència de l'anàlisi. Si és massa lenta se li poden escapar més gotes, i com a conseqüència tardar més a fer la detecció.

- Els productes més viscosos com ha estat en el cas de l'oli lubricant, es detecten amb molta eficiència, i en l'anàlisi es generen uns contorns molt ben definits.

- La càmera de vídeo full HD fins 1920 x 1080 ofereix la qualitat suficient per capturar qualsevol tipus de detall. Ha quedat molt clar al haver superat les proves amb la fuita d'aigua.

- La interfície és agradable, molt intuïtiva i fàcil d'utilitzar.

- Tenim una aplicació escalable i capaç d'incorporar moltes noves funcionalitats en les futures versions.

Es considera que s'han assolit els objectius plantejats inicialment. Un dels objectius que es podria dir que encara no s'ha validat del tot, és el de testejar l'aplicació amb el màxim tipus de productes: oli tèrmic o lubricant, tinta, cola, aigua, etc. Això no suposa cap problema perquè ja s'ha provat de manera molt real amb un oli lubricant que va adquirint diferents estats durant la fuita (de més transparents a més opacs) i per tant ja ens assegura que funcionarà amb productes més opacs. A més l'altre prova, i també amb èxit, ha estat amb el producte més transparent de tots, que és l'aigua, i això ja ens assegura que funcionarà amb qualsevol producte.

Respecte el punt que volia controlar els moviments i el *zoom* de la càmera remotament, es va investigar i buscar productes existents en el mercat. Cap de les càmeres IP adquirides i que tenen servomotor per fer tots els moviments de manera remota, tenen el protocol obert. Són càmeres que només permeten ser controlades mitjançant l'aplicació pròpia del fabricant i aquest no ofereix, ni les llibreries, ni el protocol.

La planificació marcada a l'inici del treball, s'ha seguit seguit sense problemes, i la metodologia prevista ha estat la adequada, degut a que es tenien les idees bastant clares. L'únic imprevist, va ser el de la càmera controlada remotament, però es va decidir deixar-ho per una futura fase i es va anar directe a la base real del projecte.

6.2 Línies de treball futur

Es considera que les línies de futur és una part important d'aquest projecte, donat que es tracta d'un projecte viu, i amb projecció de créixer i tenir la possibilitat de ser útil.

S'hi pot adaptar fàcilment una càmera de visió nocturna o una càmera tèrmica (infraroja) per poder operar també sense llum ambient. La de visió nocturna és pot trobar per preus molt més assequibles que la tèrmica, i per aquesta raó serà prioritària. Llavors es farà un seguiment dels preus i característiques de les càmeres tèrmiques, les que ja estan en el mercat i les novetats. Quan se'n trobi una a un bon preu, se'n considerarà la seva implementació.

Treballar amb imatges de visió nocturna i tèrmiques serà com un nou projecte dins un projecte ja existent, donat que treballen amb formats diferents. En el cas de les imatges tèrmiques, s'hauran de sobreposar a les imatges normals.

Una altre funcionalitat a afegir, és la que ja era un dels objectius inicials, la de incorporar el control remot d'una càmera IP. S'ha de seguir buscant fins trobar una càmera que tingui el protocol PTZ (Pan Tilt Zoom) obert, totalment a disposició de l'usuari. D'aquesta manera es podran implementar els controls a l'apart d'ajustos de l'aplicació, fent més fàcil la localització dels punts a analitzar. Es podran memoritzar totes les posicions juntament amb les dades de les zones de vigilància.

També seria una molt bona millora afegir una o vàries imatges de la fuita en el correu electrònic que avisa a l'usuari d'una possible incidència. Així aquest, podria verificar al moment el seu abast.

I finalment, arrel de tenir un control remot de la càmera, es podria començar a plantejar la implementació d'un algoritme SIFT o SURF (apartat 2.2.5). Aquests algoritmes detecten i descriuen les característiques locals, i estan pensats per aconseguir la invariància de la informació davant l'escalat, canvi d'orientació, canvi de la il·luminació, etc. Podrien localitzar automàticament vàlvules, unions, canonades, etc.

7. Glossari

Algoritme: Conjunt prescrit d'instruccions o regles ben definides, ordenades i finites que permet realitzar una activitat mitjançant passos successius que no generen dubtes a qui ha de realitzar l'activitat.

Histograma: En una imatge, és la representació gràfica o analítica de la distribució relativa de cada valor possible de píxel d'imatge.

Llibreria: En informàtica, una llibreria és un conjunt d'implementacions funcionals, codificades en un llenguatge de programació, que ofereix una interfície en definida per la funcionalitat que s'invoca.

Mostrejar: És el primer procés de digitalització que pateix un senyal. És una operació que consisteix a obtenir mostres espaiades en temps o en espai d'un senyal analògic.

Píxel: L'element bàsic d'una imatge. Ve de l'abreviatura de les paraules angleses "Picture element"; de aplicabilitat individual un color o intensitat que es pot diferenciar dels altres mitjançant un procediment determinat.

Prototip: Des del punt de vista de la informàtica, un prototip de un sistema, és una mostra més simplificada d'aquest.

Quantificar: En *processament digital de senyals*, la quantificació és la discretització d'un rang continu d'amplituds per arrodoniment o truncament de valors.

Subtracció: Operació que té per objecte sostreure una quantitat d'una altra de major, el resultat de la qual és una quantitat que sumada amb el subtrahend dóna el minuend

Tecnologia : És el conjunt de coneixements tècnics, ordenats científicament, que permet dissenyar i crear bens i serveis que faciliten l'adaptació al medi ambient i satisfer tant les necessitats essencials com els desitjos de la humanitat.

Vàlvula: És un dispositiu mecànic amb el qual es pot iniciar, aturar o regular la circulació (pas) de líquids o gasos mitjançant una peça movible que obre, tanca o obstrueix de forma parcial un o més orificis o conductes.

8. Bibliografia

[1] **Anguita, J.** (2011). “Definición de la fleografía”. *La flexografia de de alta calidad* (1ª Edición) Llinars del Valles: Technologic Tapes, S.L.

[2] **MathWorks**. Computer Vision System Toolbox [en línia] [Data de consulta: 5 d’octubre de 2018] Disponible a: https://www.mathworks.com/products/computer-vision.html

[3] **American Petroleum Institute** (2002). “Introduction”. *Computional Pipeline Monitoring for Liquid Pipelines* (Second Edition) Washington, D.C: API publishing services.

[4] **American Petroleum Institute** (2002). “Technical Overview”. *Computional Pipeline Monitoring for Liquid Pipelines* (Second Edition) Washington, D.C: API publishing services.

[5] **Wikipedia**. Leak detection [en línia] [Data de consulta: 8 d’octubre de 2018] Disponible a: https://en.wikipedia.org/wiki/Leak_detection

[6] **Geiger, G.** (2006). “TRFL”. *State of the art in Leak Dectection and Localization* Germany: University of Applied Sciences Gelsenkirchen

[7] **Intelliview - Intelligent Vision Solutions**. Intelliviewew Products [en línia] [Data de consulta: 11 d’octubre de 2018] Disponible a: https://intelliviewtech.com/solutions/video-analytics/

[8] **Intelliview - Intelligent Vision Solutions**. Intelliviewew Products [en línia] [Data de consulta: 11 d’octubre de 2018] Disponible a: https://intelliviewtech.com/solutions/industrial-fluid-leak/

[9] **Cognex** ,.Artificial Vison Products [en línia] [Data de consulta: 13 d’octubre de 2018] Disponibe a: https://www.cognex.com/es-cl/what-is/machine-vision/benefits

[10] **Iriziar Ligthning the road.** Magiceye product [en línea] [Data de consulta: 13 d’octubre de 2018] http://www.irizar.com/en/sistemas-de-seguridad-con-tecnologia-irizar-para-los-pasajeros/

[11] **Pujalte Piñán, S.** (2009). “Introductió al processament d’imatges”. *Codificació del so i de la imatge* (Primera edició) Barcelona: Eureca Media, SL (FUOC)

[12] **Pajares Martinsanz, G. de la Cruz Gacía, J M.** (2007). “Visión Artificial”. *Visión por Computador* (2ª Edición) Madrid: RA-MA

[13] **Departamento informática.** (2017). “Processamiento en el dominio espacial”. *Processamiento de imágenes digitales* Sevilla : Universidad de Sevilla

[14] **Pujalte Piñán, S.** (2009). “Filtratge lineal i no lineal”. *Codificació del so de la imatge* (Primera edició) Barcelona: Eureca Media, SL (FUOC)

[15] **Pajares Martinsanz, G. de la Cruz Gacía, J M.** (2007). “Extracción de bordes, esquinas y puntós de interés”. *Visión por Computador* (2ª Edición) Madrid: RA-MA

[16] **Pajares Martinsanz, G. de la Cruz Gacía, J M.** (2007). “Descripción de líneas y contronos”. *Visión por Computador* (2ª Edición) Madrid: RA-MA

[17] **Opencv dev team** (2011-2014). *OpenCV Home* [en línia] OpenCV [Data de consulta: 8 d'Octubre 2018] Disponible a: https://opencv.org

[18] **Aforget.NET dev team** (2017-2018). Aforget.NET Framework [en línia] [Data de consulta: 10 d’octubre de 2018] http://www.aforgenet.com/

[19] **Blog SEAS** (2018). Librerias de vision artificial [en línia] SEAS [Data de consulta: 12 d’octubre de 2018] https://www.seas.es/blog/seas-cursos-online/librerias-de-vision-artificial/

[20] **Logitech suport t** (2018). HD Prp Webcam C920 Support [en línia] Logitech[Data de consulta: 22 d’octubre de 2018] https://support.logitech.com/en_us/product/hd-pro-webcam-c920

[21] **Pajares Martinsanz, G. de la Cruz Gacía, J M.** (2007). “Suavizado, realzado y correcciones radiométricas”. *Visión por Computador* (2ª Edición) Madrid: RA-MA

[22] **Pajares Martinsanz, G. de la Cruz Gacía, J M.** (2007). “Extracción de bordes, esquinas y puntós de interés”. *Visión por Computador* (2ª Edición) Madrid: RA-MA

[23] **Kaehler, A. Bradski, G.** (2016). “Contours”. *Learning OpenCV3* (First edition) Seastopol: O’REAILLY

[24] **Kaehler, A. Bradski, G.** (2016). “Filters and Convolution”. *Learning OpenCV3* (First edition) Seastopol: O’REAILLY

[25] **Wikipedia** (2018). Common Language Runtime [en línia] Data de consulta: 19 de Novembre de 2018]
https://es.wikipedia.org/wiki/Common_Language_Runtime

[26] **Microsoft** (2018). Información general acerca de .NET [en línia] Data de consulta: 10 de Desembre de 2018] https://docs.microsoft.com/es-es/dotnet/framework/get-started/overview

9. Annexos

A 1. Codi de l'anàlisi dels modes de funcionament

En aquest apartat s'inclou el codi ubicat dins de la instrucció *switch* de l'esdeveniment *tick* del control *timer*, i que s'encarrega de l'anàlisi del sistema. Per tant, la part més important de l'aplicació.

```
switch (mode) {

case 1: // Mode 1: Direct boder detecction
       //Get the mashed image
       imgCannyOnlyWatchingZone.copyTo(toFindContourns);

       //Figure out max area
       if (activeSecondWatchingZone == true){

              if ((watchingZone.width*watchingZone.height) <
              (watchingZone2.width*watchingZone2.height)) {

                     maxArea = (watchingZone.width*watchingZone.height) * 0.6;
              }
              else {
                     maxArea = (watchingZone2.width*watchingZone2.height) * 0.6;
              }
       }
       else {
              maxArea = (watchingZone.width*watchingZone.height) * 0.6;
       }

       ////Locate the new movements on canny image
       boundingRectCreation(toFindContourns, contours, largest_contour_index,
       bounding_rect, maxArea, resetContourns,1);

       //Convert to get colors for rectangles
       cv::cvtColor(imgGrayscale, imgAnalysis, COLOR_GRAY2BGRA);
       imgAnalized = imgAnalysis.clone();


       if ((bounding_rect.x > watchingZone.x) && ((bounding_rect.x +
       bounding_rect.width) < (watchingZone.x + watchingZone.width))&&
       (bounding_rect.y > watchingZone.y) && ((bounding_rect.y +
       bounding_rect.height) < (watchingZone.y + watchingZone.height)))
       {

           drawContours(imgAnalized, contours, largest_contour_index,
           colorMovementZone, CV_FILLED, 8, hierarchy);
           //rectangle(Mat& img, Point pt1, Point pt2, const Scalar& color, int
           thickness, int lineType, int shift)
              rectangle(imgAnalized, bounding_rect, Scalar(0, 0, 255), 3, 8, 0);
              notification();
       }

       else {
              rectangle(imgAnalized, bounding_rect, Scalar(255, 0, 0), 3, 8, 0);
              resetContourns = true;
       }
```

```
        if (activeSecondWatchingZone == true) {

            if ((bounding_rect.x > watchingZone2.x) && ((bounding_rect.x +
            bounding_rect.width) < (watchingZone2.x + watchingZone2.width))
               && (bounding_rect.y > watchingZone2.y) && ((bounding_rect.y +
            bounding_rect.height) < (watchingZone2.y + watchingZone2.height)))
               {
               drawContours(imgAnalized, contours, largest_contour_index,
               colorMovementZone, CV_FILLED, 8, hierarchy);

               rectangle(imgAnalized, bounding_rect, Scalar(0, 0, 255), 3, 8, 0);
               notification();
               }
        }

        //Show Watching zone
        rectangle(imgAnalized, watchingZone, Scalar(0, 255, 0), 2, 8, 0);

        if (activeSecondWatchingZone == true)  {
               rectangle(imgAnalized, watchingZone2, Scalar(0, 255, 0), 2, 8, 0);
        }

        imgAnalized.copyTo(imgAnalysis);

        //Show to picture box
        imgGrayscaleToShowHBitMap = CreateBitmap(imgAnalysis.cols, imgAnalysis.rows,
        1, 32, imgAnalysis.data);
        imgGrayscaleToShowBitMap =
        Bitmap::FromHbitmap((IntPtr)imgGrayscaleToShowHBitMap);
        picRealTime->Image = imgGrayscaleToShowBitMap;

        cvtColor(toFindContourns, imgCannyToShow, COLOR_GRAY2BGRA);

        imgCannyToShowHBitMap = CreateBitmap(imgCannyToShow.cols,
        imgCannyToShow.rows, 1, 32, imgCannyToShow.data);
        imgCannyToShowBitMap = Bitmap::FromHbitmap((IntPtr)imgCannyToShowHBitMap);
        picCannyRealTime->Image = imgCannyToShowBitMap;

        break;


case 2://Mode 2: Still Master with real time

        if (comparationBaseOnColor == false) {
            // convert to grayscale
            cv::cvtColor(imgGrayscale, imgAnalysis, COLOR_GRAY2BGRA);
            cv::cvtColor(imgMaster, imgMaster, CV_BGR2GRAY);
            //Substract images
            subtract(imgGrayscale, imgMaster, imgResult);
            //Apply Threshold and Morphology in order remove backgorung noise

            if (skipThresholdOnSubbstraction == false) {
               threshold(imgResult, imgResult, 15, 255, CV_THRESH_BINARY);
            }
            if (skipMorphologyOnSubbstraction == false) {
            morphologyEx(imgResult, imgResult, CV_MOP_OPEN, _kernel, cv::Point2i(-1,
            -1), 1);
            }
```

```
    //Image for countors detecting ang generate reactangle for overlaping to
    realtime

    imgResult.copyTo(toFindContourns);
    cv::cvtColor(imgResult, imgResult, COLOR_GRAY2BGRA);
       }
else {
    // convert to color
    cv::cvtColor(imgOriginalContrastBrightness, imgAnalysis, COLOR_BGR2BGRA);
    cvtColor(imgMaster, imgMaster, COLOR_BGR2BGRA);
    //Substract images
    subtract(imgAnalysis, imgMaster, imgResult);
    //Apply Threshold and Morphology in order remove backgorung noise
    if (skipThresholdOnSubbstraction == false) {
       threshold(imgResult, imgResult, 15, 255, CV_THRESH_BINARY);
    }
    if (skipMorphologyOnSubbstraction == false) {
    morphologyEx(imgResult, imgResult, CV_MOP_OPEN, _kernel, cv::Point2i(-1,
    -1), 1);
    }
       if (FinalGreyScaleConvertion == true){
           //Final grey scale convertion to base on color
           cv::cvtColor(imgResult, imgResult, CV_BGR2GRAY);
           imgResult.copyTo(toFindContourns);
           cv::cvtColor(imgResult, imgResult, COLOR_GRAY2BGRA);
       }
       else {
           cv::cvtColor(imgResult, imgResult, COLOR_BGR2BGRA);
       }
}


//Locate the new movements on result image

boundingRectCreation(toFindContourns, contours, largest_contour_index,
bounding_rect, 15000, resetContourns,2);

imgAnalized =imgAnalysis.clone();


if ((bounding_rect.x > watchingZone.x) && ((bounding_rect.x +
bounding_rect.width) < (watchingZone.x + watchingZone.width))&&
(bounding_rect.y > watchingZone.y) && ((bounding_rect.y +
bounding_rect.height) < (watchingZone.y + watchingZone.height)))
{
    drawContours(imgAnalized, contours, largest_contour_index,
    colorMovementZone, CV_FILLED, 8, hierarchy);

    rectangle(imgAnalized, bounding_rect, Scalar(0, 0, 255), 3, 8, 0);
    notification();
}
else {
    rectangle(imgAnalized, bounding_rect, Scalar(255, 0, 0), 3, 8, 0);
    resetContourns = true;
}
if (activeSecondWatchingZone == true)
{
    if ((bounding_rect.x > watchingZone2.x) && ((bounding_rect.x +
    bounding_rect.width) < (watchingZone2.x + watchingZone2.width))&&
    (bounding_rect.y > watchingZone2.y) && ((bounding_rect.y +
    bounding_rect.height) < (watchingZone2.y + watchingZone2.height)))
    {
```

```
                drawContours(imgAnalized, contours, largest_contour_index,
                colorMovementZone, CV_FILLED, 8, hierarchy);

                rectangle(imgAnalized, bounding_rect, Scalar(0, 0, 255), 3, 8, 0);
                notification();
                }
        }

        //Show Watching zone
        rectangle(imgAnalized, watchingZone, Scalar(0, 255, 0), 2, 8, 0);

        if (activeSecondWatchingZone == true) {
                rectangle(imgAnalized, watchingZone2, Scalar(0, 255, 0), 2, 8, 0);
        }

        imgAnalized.copyTo(imgAnalysis);

        //Show Realtime on analysis box in grey scale or Color
        imgAnalysisToShowHBitMap = CreateBitmap(imgAnalysis.cols, imgAnalysis.rows,
        1, 32, imgAnalysis.data);
        imgAnalysisToShowBitMap =
        Bitmap::FromHbitmap((IntPtr)imgAnalysisToShowHBitMap);
        this->picRealTime->Image = imgAnalysisToShowBitMap;



case 3://Mode 3: Master delayed with realtime

        //Automaster control
        counterForDelayControl = counterForDelayControl + 1;

        if (counterForDelayControl > counterForDelayControlToCompare)
        {
           counterForDelayControlToCompare = counterForDelayControlToCompare +
        delayInterval;
           actualFrame.copyTo(previousFrame);
           skipSubstraction = true;
           this->picMaster->Image = this->picCamera->Image;
        }

        if (skipSubstraction == true) {//For avoiding equal actual and previous frame
           skipSubstraction = false;
        }
        else {
           subtract(actualFrame, previousFrame, imgResult);

           if (skipThresholdOnSubbstraction == false)
           {
               threshold(imgResult, imgResult, 15, 255, CV_THRESH_BINARY);
           }
            if (skipMorphologyOnSubbstraction == false)
            {
            morphologyEx(imgResult, imgResult, CV_MOP_OPEN, _kernel, cv::Point2i(-1,
            -1), 1);
            }
        }


        if (comparationBaseOnColor == false) {// convert to grayscale
               cv::cvtColor(actualFrame, actualFrame, COLOR_GRAY2BGRA);
        //Image for countors detecting ang generate reactangle for overlaping to
        realtime
```

```
imgResult.copyTo(toFindContourns);
cv::cvtColor(imgResult, imgResult, COLOR_GRAY2BGRA);
}
else {// convert to Color
	if (FinalGreyScaleConvertion == true) { //Final grey convertion base
	on color
	cv::cvtColor(imgResult, imgResult, CV_BGR2GRAY);
	imgResult.copyTo(toFindContourns);
	cv::cvtColor(imgResult, imgResult, COLOR_GRAY2BGRA);
	}
	else {
		cv::cvtColor(imgResult, imgResult, COLOR_BGR2BGRA);
	}
}


//Locate the new movements on result image
boundingRectCreation(toFindContourns, contours, largest_contour_index,
bounding_rect, 15000, resetContourns,3);

imgAnalized = actualFrame.clone();

if ((bounding_rect.x > watchingZone.x) && ((bounding_rect.x +
bounding_rect.width) < (watchingZone.x + watchingZone.width))&&
(bounding_rect.y > watchingZone.y) && ((bounding_rect.y +
bounding_rect.height) < (watchingZone.y + watchingZone.height)))
{
	drawContours(imgAnalized, contours, largest_contour_index,
	colorMovementZone, CV_FILLED, 8, hierarchy);

	rectangle(imgAnalized, bounding_rect, Scalar(0, 0, 255), 3, 8, 0);
	notification();
}
else {
	rectangle(imgAnalized, bounding_rect, Scalar(255, 0, 0), 3, 8, 0);
	resetContourns = true;
}

if (activeSecondWatchingZone == true)
{
    if ((bounding_rect.x > watchingZone2.x) && ((bounding_rect.x +
    bounding_rect.width) < (watchingZone2.x + watchingZone2.width))&&
    (bounding_rect.y > watchingZone2.y) && ((bounding_rect.y +
    bounding_rect.height) < (watchingZone2.y + watchingZone2.height)))
    {
       drawContours(imgAnalized, contours, largest_contour_index,
       colorMovementZone, CV_FILLED, 8, hierarchy);

       rectangle(imgAnalized, bounding_rect, Scalar(0, 0, 255), 3, 8, 0);
       notification();
    }
}


//Show Watching zone
rectangle(imgAnalized, watchingZone, Scalar(0, 255, 0), 2, 8, 0);

if (activeSecondWatchingZone == true) {
	rectangle(imgAnalized, watchingZone2, Scalar(0, 255, 0), 2, 8, 0);
}
```

```
imgAnalized.copyTo(imgAnalysis);

//Show Realtime on analysis box in grey scale or Color
imgAnalysisToShowHBitMap = CreateBitmap(imgAnalysis.cols, imgAnalysis.rows,
1, 32, imgAnalysis.data);
imgAnalysisToShowBitMap =
Bitmap::FromHbitmap((IntPtr)imgAnalysisToShowHBitMap);
this->picRealTime->Image = imgAnalysisToShowBitMap;

//Show the result of substraction once processed in grey or Color
delayMasterToShowHBitMap = CreateBitmap(imgResult.cols, imgResult.rows, 1,
32, imgResult.data);
delayMastertToShowBitMap =
Bitmap::FromHbitmap((IntPtr)delayMasterToShowHBitMap);
this->picRealTimeWithDelay->Image = delayMastertToShowBitMap;
//break;
```

A 2. Funcions amb el codi complet

Ajust de la brillantor i contrast de la imatge.

```
cv::Mat imageContrastBrightAdjusted (double alpha, int beta, cv::Mat image){
	cv::Mat new_image = Mat::zeros(image.size(), image.type());
	for (int y = 0; y < image.rows; y++) {
		for (int x = 0; x < image.cols; x++) {
			for (int c = 0; c < 3; c++) {
			new_image.at<Vec3b>(y, x)[c] =
		saturate_cast<uchar>(alpha*(image.at<Vec3b>(y, x)[c]) + beta);
			}
		}
	}
return new_image;
```

Escriure els valors de les zones de vigilància al fitxer de text.

```
void writeWatchingZoneParametersFromFile() {

	ofstream watchingZoneFile("c:\\OpenCV\\Data\\watchingZone.txt");

	if (watchingZoneFile.is_open())
	{
		watchingZoneFile << watchingZone.height << endl;
		watchingZoneFile << watchingZone.width << endl;
		watchingZoneFile << watchingZone.x << endl;
		watchingZoneFile << watchingZone.y << endl;

		watchingZoneFile << watchingZone2.height << endl;
		watchingZoneFile << watchingZone2.width << endl;
		watchingZoneFile << watchingZone2.x << endl;
		watchingZoneFile << watchingZone2.y << endl;
		watchingZoneFile.close();	//Close text file
		}
	}
```

Llegir els valors de les zones de vigilància del fitxer de text.

```
void readWatchingZoneParametersFromFile() {
	ifstream  watchingZoneFile("c:\\OpenCV\\Data\\watchingZone.txt");
	string line;
	if (watchingZoneFile.is_open())
	{
		getline(watchingZoneFile, line);
		stringstream(line) >> watchingZone.height;
		getline(watchingZoneFile, line);
		stringstream(line) >> watchingZone.width;
		getline(watchingZoneFile, line);
		stringstream(line) >> watchingZone.x;
		getline(watchingZoneFile, line);
		stringstream(line) >> watchingZone.y;
		getline(watchingZoneFile, line);
		stringstream(line) >> watchingZone2.height;
		getline(watchingZoneFile, line);
		stringstream(line) >> watchingZone2.width;
		getline(watchingZoneFile, line);
		stringstream(line) >> watchingZone2.x;
		getline(watchingZoneFile, line);
		stringstream(line) >> watchingZone2.y;
		watchingZoneFile.close();
	}
```

Escriure els valors dels ajustos de la imatge mestre.

```
void writeMasterAdjustmentsParametersToFile() {

	ofstream
	masterAdjustmentsFile("c:\\OpenCV\\Data\\masterAdjustments.txt");

	if (masterAdjustmentsFile.is_open())
	{
		masterAdjustmentsFile << Brightness << endl;
		masterAdjustmentsFile << Contrast << endl;
		masterAdjustmentsFile << Blurred << endl;
		masterAdjustmentsFile << MaxThreshold << endl;
		masterAdjustmentsFile << MinThreshold << endl;
		masterAdjustmentsFile.close();
	}
}
```

Llegir i assignar els valors dels ajustos de la imatge mestre.

```
void readMasterAdjustmentsParametersFromFile() {

	ifstream
	masterAdjustmentsFile("c:\\OpenCV\\Data\\masterAdjustments.txt");
	string line;

		if (masterAdjustmentsFile.is_open())
		{
			getline(masterAdjustmentsFile, line);
```

```
                        stringstream(line) >> Brightness;
                        getline(masterAdjustmentsFile, line);
                        stringstream(line) >> Contrast;
                        getline(masterAdjustmentsFile, line);
                        stringstream(line) >> Blurred;
                        getline(masterAdjustmentsFile, line);
                        stringstream(line) >> MaxThreshold;
                        getline(masterAdjustmentsFile, line);
                        stringstream(line) >> MinThreshold;
                        masterAdjustmentsFile.close();

                        this->numUpDwBlurred->Value = Blurred;
                        this->tbrBlurred->Value = Blurred;
                        this->numUpDwMaxThreshold->Value = MaxThreshold;
                        this->tbrMaxThreshold->Value = MaxThreshold;
                        this->numUpDwMinThreshold->Value = MinThreshold;
                        this->tbrMinThreshold->Value = MinThreshold;
                        this->numUpDwContrast->Value = Contrast;
                        this->tbrContrast->Value = Contrast;
                        this->numUpDwBrightness->Value = Brightness;
                        this->tbrBrightness->Value = Brightness;
                }
        }
```

Eliminar els contorns fora de les zones de vigilància.

```
cv::Mat erasePixelsOutZOneOnBinaryImg(cv::Rect watchingZone, cv::Rect
watchingZone2, cv::Mat image)
   {
   cv::Mat new_image = Mat::zeros(image.size(), image.type());

   for (int y = 0; y < image.rows; y++) {
          for (int x = 0; x < image.cols; x++) {

   if ((x > watchingZone.x) && (x  < (watchingZone.x + watchingZone.width))&& (y
   > watchingZone.y) && (y  < (watchingZone.y + watchingZone.height)))
          {
          new_image.at<uchar>(y, x) = image.at<uchar>(y, x);
          }

          if (activeSecondWatchingZone == true) {
                 if ((x > watchingZone2.x) && (x < (watchingZone2.x +
          watchingZone2.width))&& (y > watchingZone2.y) && (y < (watchingZone2.y
          + watchingZone2.height)))
          {
                 new_image.at<uchar>(y, x) = image.at<uchar>(y, x);
                 }
                 }
          }
   }
   return new_image;
   }
```

Creació del rectangle de la zona del contorn detectat.

```
void boundingRectCreation(cv::Mat imgToCheck, std::vector < std::vector <
cv::Point > > &contours,int &largest_contour_index, Rect &bounding_rect,
intmaximumCntournArea , bool &resetContourns, int mode){

	int largest_area = 0;
	vector<Vec4i> hierarchy;
	cv::findContours(imgToCheck, contours, hierarchy, CV_RETR_CCOMP,
	CV_CHAIN_APPROX_SIMPLE);

	for (size_t i = 0; i < contours.size(); i++)
	{
		double a = contourArea(contours[i], false);
		if (a > largest_area)
		{
			largest_area = a;
			largest_contour_index = i;
			bounding_rect = boundingRect(contours[i]);
		}
		if (largest_area > maximumCntournArea) {
			largest_area = 0;
			a = 0;
			contours.clear();}
	}
}
```

Notificació d'alarma de fuita mitjançant correu electrònic.

```
void notification()
{
	attempts = attempts + 1;

    if ((attempts > maxAttempts) && (notificationDone==false)){

	MailMessage^ msg = gcnew MailMessage("fanxicore@gmail.com",
	"marc.segura@comexi.com");
	msg->Subject = "Leaking Alert!!";
	msg->SubjectEncoding = System::Text::Encoding::UTF8;
	msg->Body = "Please, check your installation";
	msg->BodyEncoding = System::Text::Encoding::UTF8;
	msg->IsBodyHtml = false;
	SmtpClient^ client = gcnew SmtpClient("smtp.gmail.com");
	client->Credentials = gcnew
	System::Net::NetworkCredential("fanxicore@gmail.com",
	"1977fanxicore2018");
	client->Port = 587;
	client->Host = "smtp.gmail.com";
	client->EnableSsl = true;
	client->Send(msg);
	notificationDone = true;
	this->btnResetNotification->Visible = true;
   }
}
```