Treball de Fi de Màster

Màster en Enginyeria Industrial (MUEI)

Tècniques servo control amb DJI Tello

MEMÒRIA

18 de gener de 2021

Autor: Xavier Almendros Carmona

Director: Antoni Grau

Convocatòria: Gener de 2021

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

El control visual neix del creixement de la indústria automatitzada i la robòtica a causa del poc impacte d'aquestes indústries en aquells processos on no es poden controlar les variables ambientals per culpa de la inherent falta de capacitat dels sensors que es comercialitzen avui en dia a un preu viable. Tenint en compte aquesta necessitat en augment de l'aplicació de tècniques de control o observació visual sumat a la millora de la capacitat computacional que han patit els xips, era d'esperar que els últims anys hi hagi hagut un gran creixement d'articles científics on s'estudia el tractament d'imatges i les tècniques de control basades en informació extreta de les mateixes. Gràcies als avenços comentats s'han pogut implementar aquestes tècniques visuals a la indústria facilitant així l'automatització de moltes empreses a un preu més econòmic.

Per altra banda, la indústria del vehicle aeri no tripulat (a partir d'ara "dron") també ha patit grans avenços els darrers anys degut a diversos factors tals com la millora de potència dels microxips, la millora de capacitat de les bateries i fins i tot la millora dels processos de manufactura. Com a conseqüència hi ha gran varietat de drons disponibles al mercat a un preu molt raonable.

Al llarg d'aquest projecte es tracta el disseny des de zero d'un sistema de detecció i control autònom basat en imatge per a un dron comercial amb l'objectiu d'assolir un, o més, prototips de sistemes.
Aquest procés de disseny consta de cinc parts diferents:

- Selecció del dron: s'estudien diferents opcions disponibles al mercat.
- Tècniques servo control: es realitza un petit estudi de l'estat de l'art dels sistemes servo control així com l'explicació concreta del control basat en imatge.
- Tècniques de detecció: s'expliquen les diferents metodologies escollides per a la detecció de l'objectiu, el funcionament de cada una d'elles i l'algorisme implementat.
- Llaç de control: s'expliquen diferents formes d'actuar a l'hora de realitzar el control quan ja s'ha detectat l'objectiu.
- Resultats: Es combinen les diferents metodologies estudiades i analitzem el seu comportament.

Índex

Índex de figures

Índex de taules

Índex de Codis

1 Prefaci

1.1 Origen del projecte

Aquest treball neix de la motivació d'aprendre el llenguatge de programació *Python* combinat amb interés personal en la indústria aeronàutica. Durant el camí d'aprenentatge vaig descobrir la quantitat d'informació que es pot extreure d'una imatge mitjançant tècniques simples de visió per computador, amb baix cost computacional, així com amb tècniques més complexes de *Machine Learning* o *Deep Learning*.
El present treball també neix de constatar l'augment de reporters independents amb escassa capacitat econòmica que volen realitzar contingut audiovisual simple en moviment sense la necessitat d'un tercer.

1.2 Motivació

La principal motivació d'aquest treball és demostrar la capacitat d'aprenentatge assolida durant els anys d'universitat per aprendre i créixer en camps no directament relacionats amb el camp d'estudi propi.
A més a més, un altre factor motivant és la possibilitat de transformar aquest estudi en un prototip funcional a la realitat, allunyant-nos de les aproximacions en les simulacions i tenint en compte totes les possibles casuístiques que sorgeixen en l'aplicar un projecte a la realitat.

1.3 Requeriments previs

Per tal de dur a terme aquest treball a continuació es llisten els diferents requeriments previs necessaris:

- Llenguatge de programació *Python*
- Llibreria lliure *OpenCV*
- Tractament d'imatge
- Algoritmia
- Connexions mitjançant socket
- Càlcul avançat
- Enviar senyals de control al dron mitjançant computadora

2 Introducció

2.1 Objectius del projecte

El propòsit essencial d'aquest projecte és la realització d'un control autònom a un dron comercial mitjançant tècniques de visió per computador amb el llenguatge de programació *Python*.

Per tal de poder assolir aquest objectiu serà de crucial importància realitzar un estudi de les diferents formes de detecció d'objectes mitjançant visió per computador disponibles avui en dia així com les possibles diferents formes d'actuar un cop tenim el *target* identificat en la imatge.

És important que les tècniques usades tinguin un baix cost computacional, ja que la idea d'aquest projecte és acabar utilitzant-lo en drons equipats amb capacitat de càlcul per tal de poder-lo utilitzar sense la necessitat de tenir un dispositiu a prop. A més a més, un baix cost computacional comporta un cicle d'execució petit, o cosa que és equivalent, una freqüència alta que assegura una continuïtat en el moviment necessària per a un correcte funcionament de l'algorisme de detecció i control. Es fixa com a objectiu assolir una freqüència de cicle entre 2 i 20 Hz.

En aquest projecte la connexió es realitzarà a un PC, el qual realitzarà els càlculs pertinents per després enviar la senyal de control al dron.

2.2 Abast del projecte

L'abast d'aquest projecte és assolir un seguiment d'un dron mitjançant la informació extreta de la imatge obtinguda de la càmera del dron. Com s'ha comentat en els objectius, es realitzen les següents tasques:

- Comparativa drons disponibles al mercat amb baix cost.
- Estat de l'art de les tècniques de servo control.
- Mètodes de detecció d'objecte.
- Llaç de control per assolir el seguiment.
- Comparativa de les diferents metodologies estudiades combinades així com la discussió de resultats.

3 Dron DJI Tello

3.1 Introducció

Primer de tot s'ha realitzat un petit estudi de les diferents tipologies de drons que trobem al mercat. També s'ha investigat els diferents projectes realitzats per la comunitat amb l'objectiu d'aprendre de les diferents casuístiques.

A continuació es presenten les diferents tipologies de drons multirotor i les seves configuracions:

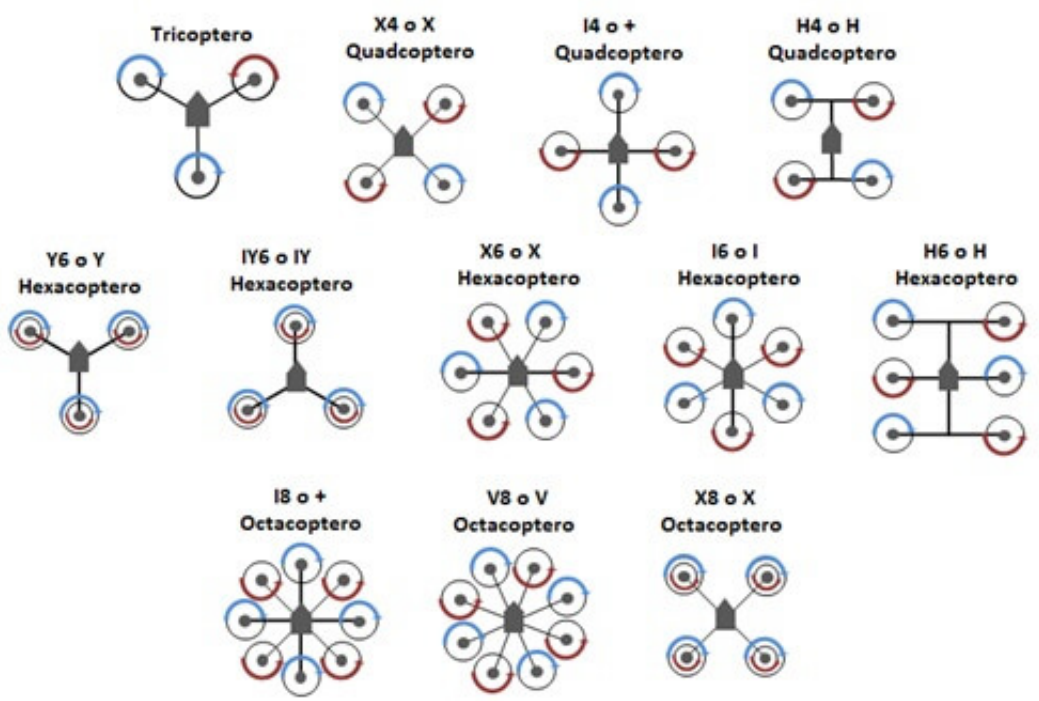


Figura 3.1: Configuracions de dron segons el nombre d'hèlixs

Existeixen moltes tipologies, però a causa de les limitacions econòmiques del projecte i a la gran oferta en el mercat de drons del tipus *"Quadcoptero X"* aquest projecte s'ha decantat per aquesta última opció.

Per introduir al lector en la dinàmica del dron, s'adjunta el següent esquema qualitatiu de com ha de ser el repartiment de potència en els rotors.

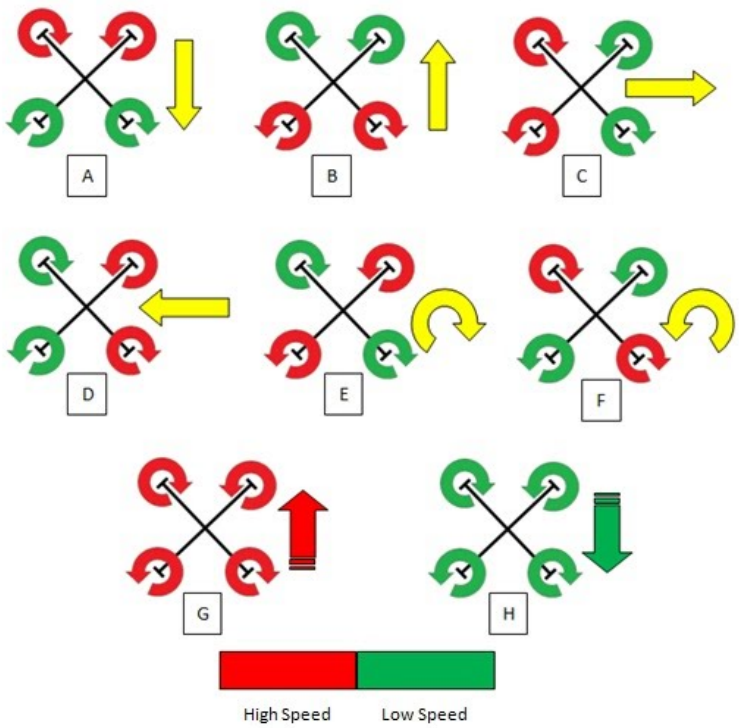


Figura 3.2: Moviment bàsics control dron

A continuació es llisten els requeriments mínims del dron per al correcte desenvolupament d'aquest projecte:

- Connexió mitjançant Wi-Fi.
- Abast 25 metres.
- Duració de la bateria mínima de 10 minuts.
- Programable en *Python*
- Configuració en "X"

Un cop seleccionada la tipologia i els requeriments mínims indispensables, fem una petita selecció dels drons disponibles al mercat amb les característiques adequades per ser utilitzats en el nostre projecte. A continuació seleccionem les millors opcions per tal de fer una petita comparació. Aquests drons consisteixen en una estructura en X amb un rotor independent a cada extrem. En funció de la potència que deriva la bateria a cada un dels rotors, aconseguim un moviment o un altre.

Robolin CoDrone

Aquesta primera opció és un dron desenvolupat per Robolink, una empresa específicament centrada en programació. En concret aquest dron permet la integració amb *Python* i Arduinio. Disposa d'un abast de 18 metres i una duració de vol de 8 minuts.

Figura 3.3: Robolink Drone

Preu: 180 €

Parrot Mambo

El següent dron és un dispositiu desenvolupat per l'empresa Parrot. També permet la programació mitjançant *Python*. El temps de vol és de 8 min i l'abast de 20 metres.

Figura 3.4: Parrot Drone

Preu: 59 €

DJI Tello Drone

Per últim, l'opció triada per a la realització del treball. El dron desenvolupat per l'empresa *Shenzhen Ryze Technologies* que incorpora tecnologia *Intel*. El temps de vol del dron és de 13 minuts i l'abast de 100 metres.

Figura 3.5: DJI Tello Drone

Preu: 120 €

Aquest dron permet una fàcil interacció mitjançant "programació en blocs"de Arduino i *Python*. Més endavant es detallen els detalls de la connexió i les característiques principals d'aquest dron.

També incorpora una SDK molt senzilla amb la qual ens podem comunicar amb el DJI Tello per tal d'enviar les accions que creiem necessàries.

L'empresa *Shenzhen Ryze Technologies* disposa de gran varietat de drons amb finalitats molt diverses. Per al nostre projecte, també ens serveix el DJI Tello Drone EDU, que és una versió millorada del nostre dron. Incorpora una SDK 2.0 amb més funcions, com el "Multi-vol". Tot i això, per als nostres algorismes ens és suficient amb la versió simple.

3.2 Característiques principals

Les característiques i dimensions d'aquest dron el fan idoni per l'aplicació del nostre projecte. Les característiques més importants per al projecte es poden veure en les taules que hi ha a continuació:

Components principals

En la següent figura podem veure les parts principals que conformen el dron.

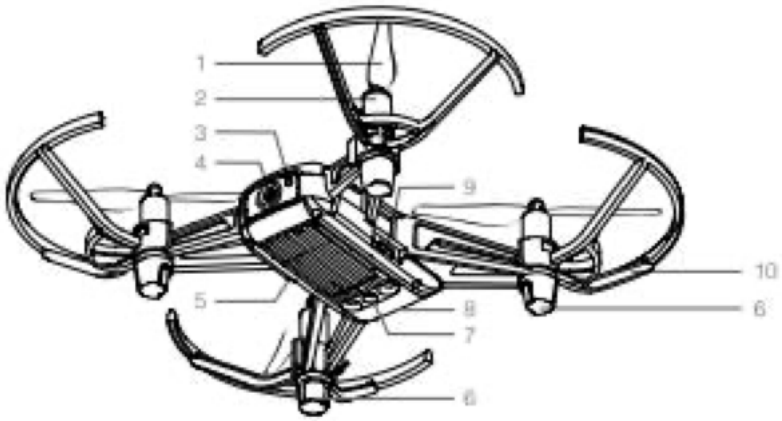


Figura 3.6: Components principals DJI Tello Dron

01. Hèlix
02. Motors
03. LED indicador d'estat
04. Càmera
05. Botó ON/OFF
06. Antenes
07. Sensor de posicionament
08. Bateria
09. Port microUSB
10. Proteccions d'Hèlix

Taula 3.1: Components principals DJI Tello Dron

Característiques principals

Nombre de rotors	4 rotors
Distància operativa màxima	100 m
Velocitat màxima	100 cm/s
Freqüència del canal de comunicació	2.4 GHz
Wi-Fi estàndar	Wi-Fi 4 (802.11n)

Taula 3.2: Característiques principals DJI Tello Drone

Ample	92 mm
Profunditat	98 mm
Altura	41 mm
Diàmetre de rotor	7.62 cm
Pes (amb Bateria)	80 g

Taula 3.3: Pes i dimensions del DJI Tello Dron

Mitjançant aquestes característiques comprovem que aquest dron compleix les exigències dinàmiques de manera còmoda. Una velocitat de 100 cm/s amb un rang d'operació de 100 m.

Característiques de la càmera

Megapíxels	5 MP
Màxima resolució de vídeo	1820x720p @ 30fps [MP4]
Format de vídeo	720p
Format de vídeo compatible	MP4
Angle de camp de visió (FOV)	82.6°

Taula 3.4: Característiques càmera DJI Tello Dron

Com es pot veure a la taula 3.4 la qualitat de vídeo que proporciona la càmera i el camp de visió són suficients per a la realització del projecte.

Característiques de la bateria

Tecnologia de la bateria	Lythium-Ion Polymer
Capacitat de la bateria	1100 mAh / 4.1 Wh
Voltatge de bateria	3.8 V
Temps de bateria	13 min

Taula 3.5: Característiques bateria DJI Tello Dron

Un punt crític d'aquest dron és l'autonomia de la bateria però amb les potències de vol(3.2) i les dimensions/pes(3.3) és difícil trobar algun altre dron amb molta millor autonomia.

3.3 Ecosistema de treball

La connexió amb el dron es realitza mitjançant Wi-Fi. És possible utilitzar qualsevol sistema operatiu per programar-lo, però aquest projecte s'ha realitzat amb un sistema *Unix*. Per enllaçar ambdós dispositius només és necessari connectar-se al dron DJI Tello com si fos una xarxa Wi-Fi oberta.

Com s'ha comentat anteriorment, és possible programar-lo en diferents llenguatges però aquest projecte es realitza amb el llenguatge *Python* degut a la seva facilitat d'aprenentatge i amplia comunitat educativa. Per tal de poder llegir la informació del dron, és a dir les imatges pel seu posterior tractat, hem de realitzar una connexió mitjançant un *Socket*:

```
UDP_IP = '192.168.10.1'
    UDP_PORT = 8889
    RESPONSE_TIMEOUT = 0.5  # s
    TIME_BTW_COMMANDS = 0.5  # s
    TIME_BTW_RC_CONTROL_COMMANDS = 0.5  # s
    last_received_command = time.time()


    VS_UDP_IP = '0.0.0.0'
```

```
    VS_UDP_PORT = 11111


    cap = None
    background_frame_read = None

    stream_on = False

    def __init__(self):

        self.address = (self.UDP_IP, self.UDP_PORT)
        self.clientSocket = socket.socket(socket.AF_INET,
                                          socket.SOCK_DGRAM)
        self.clientSocket.bind(('', self.UDP_PORT))
        self.response = None
        self.stream_on = False


        thread = threading.Thread(target=self.run_udp_receiver, args=())
        thread.daemon = True
        thread.start()
```

Codi 3.1: Connexió

Una vegada tenim realitzada aquesta connexió, ja podem enviar i rebre *packets* de dades amb el nostre dron. En aquest projecte s'utilitza per a rebre la informació de la càmera i, un cop s'ha tractat, s'envia un senyal de control amb les velocitats desitjades.

Aquestes velocitats s'envien al dron de manera numèrica en un rang de *[-100, 100] cm/s*

En el nostre cas, utilitzem una funció predissenyada que ens permet enviar les diferents velocitats en l'espai, tant de rotació com translació, al mateix temps. Gràcies a aquesta funció, el dron pot fer trajectòries diagonals o corbes i no només moviments cartesians.

Una vegada tenim calculades les diferents velocitats amb els mètodes posteriorment explicats les enviem al dron Tello mitjançant la següent funció:

```
def send_rc_control(self, left_right_velocity, forward_backward_velocity,
    up_down_velocity, yaw_velocity):

        if int(time.time() * 1000) - self.last_rc_control_sent < self.
    TIME_BTW_RC_CONTROL_COMMANDS:
            pass
        else:
            self.last_rc_control_sent = int(time.time() * 1000)
            return self.send_command_without_return('rc %s %s %s %s' % (
    left_right_velocity, forward_backward_velocity,
up_down_velocity, yaw_velocity))
```

Codi 3.2: Enviament de velocitats al dron

Aquesta funció s'executa dintre de la classe Tello que posteriorment s'inicialitza en el nostre programa principal.

Com es pot veure, també permet modificar el temps d'enviament dels *packets* d'informació. Tot i així, el fabricant recomana un mínim de 50 mil·lisegons entre paquets de dades per un correcte funcionament.

4 Control Servo-Visual

4.1 Introducció, terminologia i notació

El present treball utilitza una tecnologia de control basada en imatge. A continuació es realitza una petita introducció sobre les diferents tecnologies de control visual que podem trobar a la indústria actualment.

El control visual neix del creixement en la indústria automatitzada. La gran majoria de robots que operen actualment al món, operen en fàbriques que han estat específicament dissenyades per adaptar-se al robot. La indústria del robot ha tingut menys impacte allà on no es poden controlar les variables ambientals. Aquesta limitació és a causa de la inherent falta de capacitat dels sensors que es comercialitzen amb els robots avui dia. Està demostrat que la integració dels sensors és fonamental per incrementar la versatilitat i aplicacions dels robots però actualment el cost d'aquests sensors és massa elevat per utilitzar-se en certs camps de la robòtica industrial. Per omplir aquesta necessitat del mercat de sensors a baix cost neix la visió per computador, ja que és un sensor útil per al robot degut a que simula el sentit de la visió dels humans i permet un càlcul o mesura de l'entorn sense tocar. Factor clau en gran varietat de processos.

D'ençà que l'any 1980 el primer sistema de control visual va ser creat, el progrés ha estat molt lent, però en els últims anys s'ha vist un augment en les publicacions de recerca científica. Aquest canvi ve condicionat per l'augment de potència de càlcul computacional que permet arribar a una freqüència de càlcul suficientment acurada per poder alimentar el senyal del robot i aconseguir un moviment precís que minimitzi l'error assegurant una continuïtat en el moviment.

El control visual és la fusió resultant de diferents disciplines elementals tals com la cinemàtica, dinàmica, teoria de control, processament de la imatge i la computació a temps real. Aquesta fusió, en definitiva, té un objectiu específic que és el control d'un robot per manipular el seu entorn utilitzant la visió. Al contrari de les tècniques utilitzades abans de tant sols observació de l'entorn.

4.2 Models de projecció de càmera

S'anomena projecció tridimensional a qualsevol mètode que fa correspondre punts de l'espai sobre una superfície bidimensional. Per tal de controlar un robot usant informació d'un sistema de visió per computador és important entendre els aspectes geomètrics del processament d'imatges. Cada càmera utilitzada en el sistema crea una projecció de l'escena en funció d'on està el sensor col·locat. Aquesta projecció genera una pèrdua d'informació sobre la profunditat a la qual es troba cada punt en l'espai 3D. Com a conseqüència, és necessari extreure la informació d'una altra font. Aquesta altra font poden ser més càmeres, més punts de vista o simplement coneixent alguna relació geomètrica entre diferents punts del nostre espai 3D, com podrien ser distàncies o angles.

Seguidament fem una petita descripció dels models de projecció més utilitzats en l'actualitat per a la creació d'imatges:

- Projecció de perspectiva
  Una projecció en perspectiva es regeix pels mateixos principis que una projecció ortogràfica, explicada a continuació, però requereix que se li especifiqui un factor, en aquest cas la z, per

tal d'assegurar que els objectes més propers semblin més grans en la projecció i viceversa.

$$\pi(x, y, z) = \begin{bmatrix} \bar{u} \\ \bar{v} \end{bmatrix} = \frac{\lambda}{z} \begin{bmatrix} x \\ y \end{bmatrix} \tag{1}$$

- Projecció ortogràfica escalada
  El model ortogràfic no requereix cap factor d´escala. És un model que ignora l'efecte de la grandària d'un objecte en funció de la distància, a diferència del model de perspectiva. És un model molt utilitzat en l'enginyeria per mostrar el perfil, el detall o les mesures precises d'un objecte tridimensional. El model de projecció ortogràfica és vàlid per a aquelles situacions on la profunditat relativa dels diferents punts és petita comparat amb la distància de la càmera amb l'objecte. Per exemple, un avió o una càmera amb una distància focal (λ) gran i a diversos metres de l'objecte.

  $$\begin{bmatrix} \bar{u} \\ \bar{v} \end{bmatrix} = s \begin{bmatrix} x \\ y \end{bmatrix} \tag{2}$$

  On factor s és un escalar constant.

- Projecció Afí
  En geometria euclidiana, una transformació afí és una transformació geomètrica que preserva les línies, el paral·lelisme entre elles i la raó dels punts. No necessàriament conserva les distàncies i els angles.

  $$\begin{bmatrix} \bar{u} \\ \bar{v} \end{bmatrix} = \mathbf{A}^c \, \mathbf{P} + c \tag{3}$$

És important remarcar que per cada un dels models mencionats anteriorment, designem el centre de coordenades de la càmera amb els eixos x i y creant un pla per la imatge. L'eix z és perpendicular al pla de la imatge.

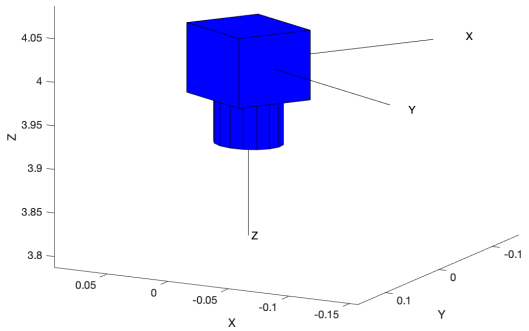


Figura 4.1: Origen de coordenades fix a la càmera representat amb *MATLAB*

L'origen de coordenades està a una distància λ del pla de la imatge. Aquesta distància s'anomena distància focal.

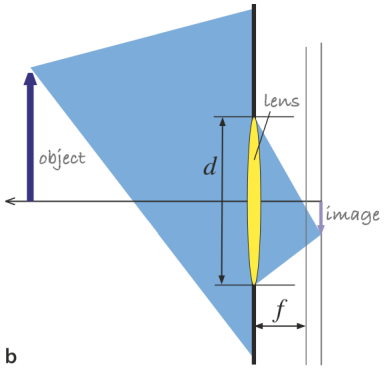


Figura 4.2: Distància focal d'una càmera

En la següent figura extreta de [1] es pot veure el marc de referència, el pla de la imatge i el punt en l'espai tridimensional tant com la seva projecció.

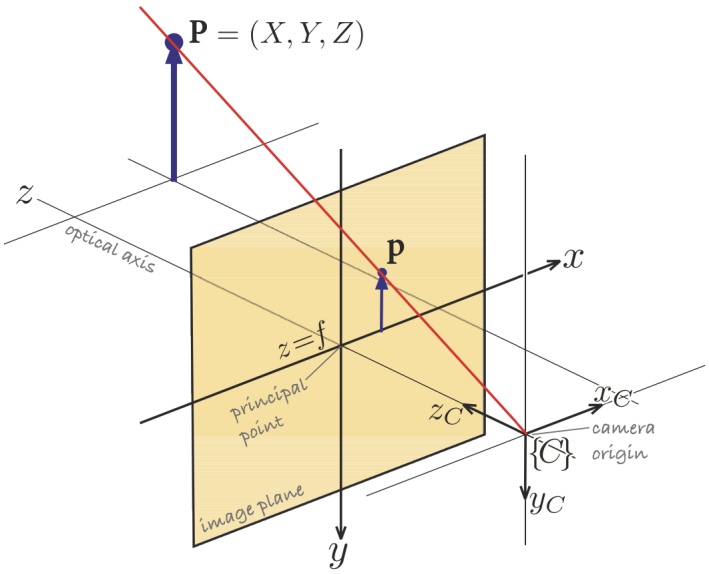


Figura 4.3: Pla de referència

4.3 Configuracions de càmera

Existeixen diferents tipus de configuració en funció del nombre de càmeres, però a la pràctica els sistemes servo-visuals utilitzen principalment dos tipus de configuracions. Pel nostre cas en el qual només tenim una càmera ho simplifiquem en dues tipologies diferents de configuració.

En primer lloc, la configuració *Eye in hand* que consisteix en muntar la càmera a l'efector terminal del robot. Un avantatge és que la distància es manté constant entre la càmera i l'efector final. En aquest cas l'objectiu és un objecte que no forma part del robot. La segona configuració consisteix en tenir la càmera fixada a l'espai de treball. En contrast amb la configuració *"Eye in hand"* tens més d'un objectiu, ja que has de controlar l'objecte al mateix temps que la posició del robot. Un altra complicació és que la velocitat final que vols controlar és la del robot i no la de la càmera. Això implica haver de fer les transformacions necessàries. Com a conseqüència

existeix un major cost computacional i una major propagació d'errors.

En la següent figura extreta de [2] es poden veure aquestes configuracions de manera gràfica i molt simplificada per entendre el concepte:

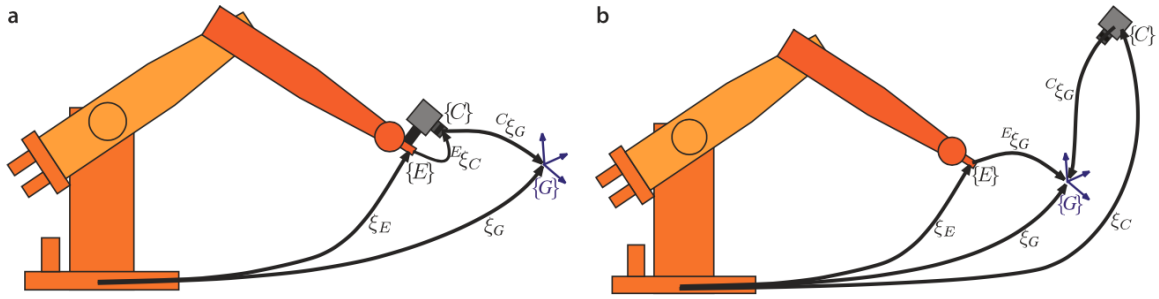


Figura 4.4: Configuracions de càmera *Eye in hand* (a) i Fixe (b)

Per al nostre projecte, com consisteix en un dron amb una càmera muntada utilitzem una configuració *Eye in hand*.

4.4 Mètodes de control

En l'actualitat ens podem trobar diferents arquitectures de control implementades en la indústria. Les més importants són les següents:

- Control basat en posició
- Control basat en imatge

En les següents figures extretes de [1] es pot veure una representació gràfica de les diferents estructures de control.

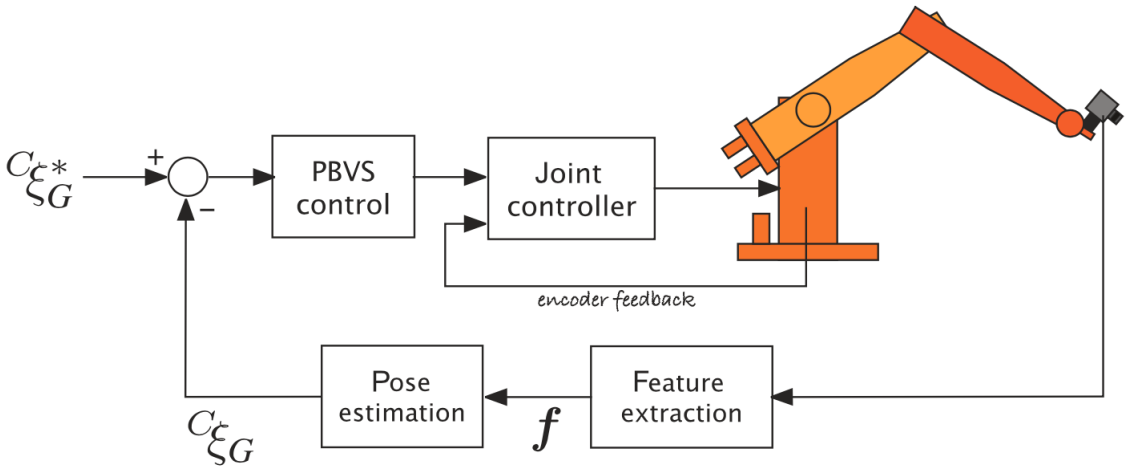


Figura 4.5: Llaç de control sistema PBVS

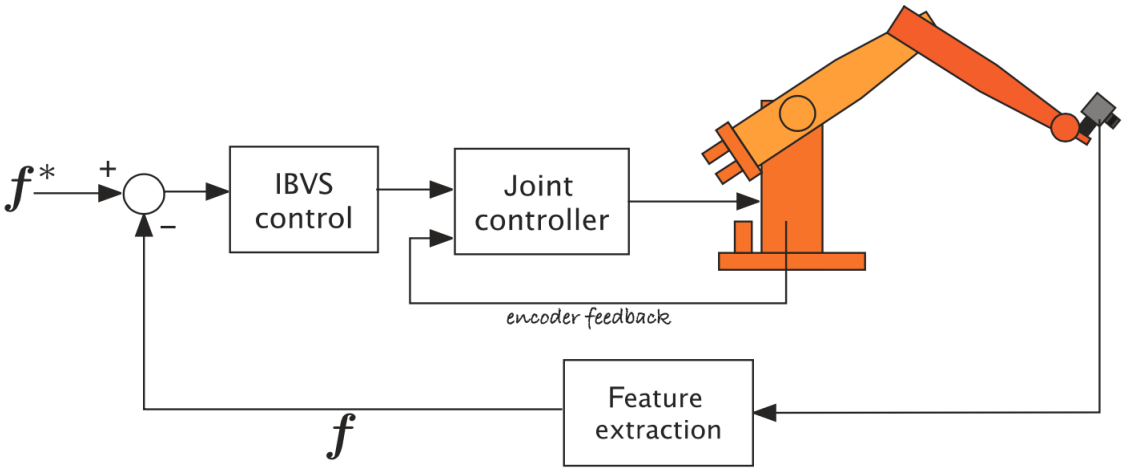


Figura 4.6: Llaç de control sistema IBVS

La principal diferència entre el control realitzat per l'arquitectura basada en imatge i l'arquitectura basada en posició és que en el primer, es calcula a partir de paràmetres directament extrets de la imatge mentre que per al control basat en posició, un cop s'ha extret la informació de la imatge primer es transforma a termes de posició per després realitzar el control.

En aquest projecte s'utilitza el control basat en imatge. A continuació s'explica més detalladament en què consisteix aquest mètode i com es desenvolupa.

4.4.1 Control basat en imatge

Com s'ha comentat en la introducció anterior, en el control visual basat en imatge el senyal de control es defineix directament en termes de paràmetres de la imatge, en contrast amb el control basat en posició que defineixen el senyal d'error en l'espai de treball tridimensional.

Com s'ha descrit a la secció 4.3 el sistema pot utilitzar una configuració de càmera fixe o *Eye in Hand*. En ambdós casos, un moviment en el manipulador genera canvis en la imatge extreta pel sistema de visió. En conseqüència, per realitzar un control basat en imatge és determinant definir una funció d'error $\boldsymbol{e}$, en paràmetres d'imatge, tal que sigui 0 quan la tasca és realitzada.

Aquesta funció d'error es pot definir utilitzant transformacions i les equacions de projeccions abans descrites o mitjançant una tècnica coneguda com a *teach by showing* que consisteix en apropar el robot a la posició desitjada per extreure, a partir de la imatge, un vector dels paràmetres desitjats amb l'objectiu de definir una funció d'error.

Tot i que l'error, $\boldsymbol{e}$, està definit en paràmetres extretes directament de la imatge, el control del robot es realitza basat en coordenades espacials tridimensionals. En conseqüència és important trobar la relació entre els canvis de posició dels paràmetres en la imatge amb la posició en l'espai de treball. A l'encarregat de parametritzar aquesta relació se'l coneix com el *Jacobià de la imatge*.

4.4.1.1 Jacobià d'una imatge El jacobià d'una imatge és un terme introduït per *Lee E. Weiss* com *feature sensity matrix*. La relació proporcionada pel *Jacobià* descriu com canvien els paràmetres de la imatge respecte a un moviment de la càmera.

Definim $\boldsymbol{r}$ com les coordenades de l'efector final del robot, en el nostre cas el propi robot, en l'espai tridimensional i $\dot{\boldsymbol{r}}$ com la seva velocitat. A continuació definim $\boldsymbol{f}$ com un vector representatiu dels paràmetres de la imatge desitjats en funció de la funció d'error prèviament definida i $\dot{\boldsymbol{f}}$ el definim com la velocitat de canvi d'aquests paràmetres en el pla de la imatge. El *Jacobià* es defineix com una transformació lineal de l'espai de treball tridimensional al pla de la imatge:

$$T : \mathcal{R}^3 \rightarrow \mathcal{R}^2 \tag{4}$$

Concretament:

$$\dot{\boldsymbol{f}} = \boldsymbol{J}_v(\boldsymbol{r})\dot{\boldsymbol{r}} \tag{5}$$

on $\boldsymbol{J}_v(\boldsymbol{r}) \in \mathbb{R}^{k\times m}$:

$$\boldsymbol{J}_v(\boldsymbol{r}) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{r}} = \begin{bmatrix} \frac{\partial f_1}{\partial r_1} & \cdots & \frac{\partial f_1}{\partial r_m} \\ \vdots & & \vdots \\ \frac{\partial f_k}{\partial r_1} & \cdots & \frac{\partial f_k}{\partial r_m} \end{bmatrix} \tag{6}$$

Per tal de deduir el *Jacobià* en el context que ens trobem, definim un punt $P = (X, Y, Z)$ en l'espai de treball respecte a la càmera. Si definim la velocitat de la càmera com $\dot{\boldsymbol{r}} = (\boldsymbol{v}, \boldsymbol{w})$ on $\dot{\boldsymbol{r}} \in \mathbb{R}^6$:

$$\dot{\boldsymbol{r}} = \begin{bmatrix} v_x, & v_y, & v_z, & w_x, & w_y, & w_z \end{bmatrix}^T \tag{7}$$

i definim $\dot{\boldsymbol{f}} \in \mathbb{R}^2$:

$$\dot{\boldsymbol{f}} = \begin{bmatrix} \dot{u}, & \dot{v} \end{bmatrix}^T \tag{8}$$

on u i v representen la projecció d'un punt (P) en el pla de la imatge.

La velocitat del punt P respecte a la càmera és:

$$\dot{\boldsymbol{P}} = -\boldsymbol{w} \times \boldsymbol{P} - \boldsymbol{v} \tag{9}$$

que reescrita de manera escalar és:

$$\begin{aligned} \dot{X} &= Y w_z - Z w_y - v_x \\ \dot{Y} &= Z w_x - X w_z - v_y \\ \dot{Z} &= X w_y - Y w_x - v_z \end{aligned} \tag{10}$$

Utilitzant l'equació 1 podem traslladar aquestes equacions al pla de referència de la imatge normalitzat ($\lambda = 1$).

$$x = \frac{X}{Z}, \quad y = \frac{Y}{Z} \tag{11}$$

La seva derivada temporal és:

$$\dot{x} = \frac{\dot{X}Z - X\dot{Z}}{Z^2}, \ \dot{y} = \frac{\dot{Y}Z - Y\dot{Z}}{Z^2} \tag{12}$$

A més a més sabem que les coordenades del pla de referència estan relacionades amb els píxels mitjançant la següent equació descomposta de 1:

$$u = \lambda' x + u_0; \quad v = \lambda' y + v_0 \tag{13}$$

on $\bar{u} = u - u_0$ i $\bar{v} = v - v_0$ són les coordenades en píxels respecte a el punt de projecció. En la següent figura podem veure aquest punt de projecció per entendre millor aquesta transformació lineal.

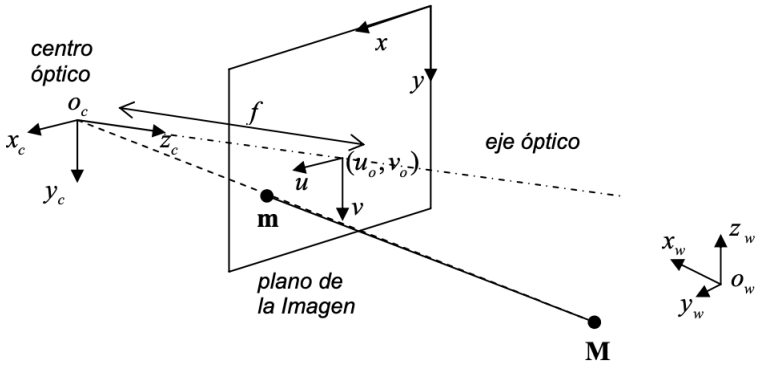


Figura 4.7: Pla de referència amb coordenades

Finalment, seguint els passos descrits prèviament i ordenant les equacions en forma matricial podem trobar la relació entre $\dot{\boldsymbol{f}}$ i $\dot{\boldsymbol{r}}$. Com hem definit en l'equació 6 a aquesta relació li diem *Jacobià*. Es pot representar de la següent manera [3]:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} \frac{\lambda'}{Z} & 0 & \frac{-\bar{u}}{Z} & \frac{-\bar{u}\bar{v}}{\lambda'} & -\frac{\lambda'^2+\bar{u}^2}{\lambda'} & -\bar{v} \\ 0 & \frac{\lambda'}{Z} & \frac{-\bar{v}}{Z} & \frac{-\lambda'^2-\bar{v}^2}{\lambda'} & -\frac{\bar{u}\bar{v}}{\lambda'} & \bar{u} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ w_x \\ w_y \\ w_z \end{bmatrix} \tag{14}$$

Realment, en el control visual basat en imatge l'objectiu és l'invers al proporcionat pel *Jacobià*, ja que estem interessats en determinar la velocitat del manipulador requerida per aconseguir un valor desitjat de $\dot{\boldsymbol{f}}$. Es proposa una solució a l'apartat 4.4.1.3.

Com es pot veure a [2] el *Jacobià* també es pot aplicar en coordenades polars o línies entre d'altres.

4.4.1.2 Problemes de percepció o ambigüitat Per a una determinada velocitat de càmera, la velocitat del punt és una funció de les coordenades d'aquest punt, la profunditat i dels paràmetres de la càmera. Cada columna del *Jacobià* indica la velocitat unitat d'un punt en el pla de la imatge causat per la component corresponent del vector velocitat de la càmera.

Mitjançant *MATLAB* podem crear diferents figures per veure com es representa el flux òptic d'una xarxa de diferents punts en l'espai global. El flux òptic es defineix com el patró de moviment aparent que generen els objectes en una escena causat pel moviment relatiu entre l'observador i l'escena. Calculat mitjançant l'equació 14.

Un exemple de flux òptic per un moviment relatiu en l'eix x entre observador i escena per uns determinats paràmetres de càmera podria ser:

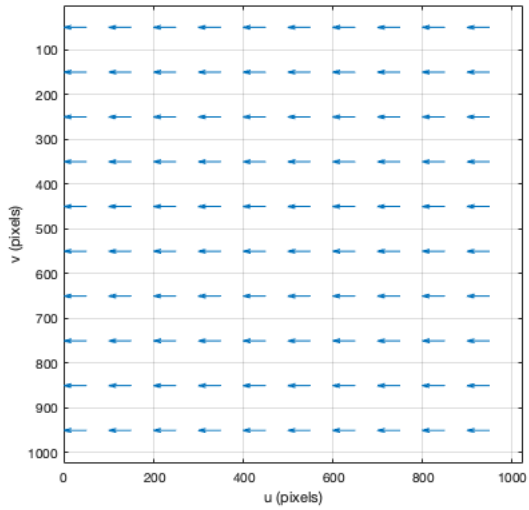


Figura 4.8: Flux òptic d'un moviment relatiu de translació *eix X*

Com és d'esperar, moure la càmera en direcció positiva en l'eix x causa un moviment dels punts en l'imatge en direcció oposada.

Pel fet que el flux òptic depèn de diferents paràmetres, això pot portar a problemes de perceptibilitat o ambigüitat. És a dir, que no sapiguem quin moviment està realitzant la nostra càmera a partir del flux òptic, ja que no el podem diferenciar de manera clara d'un altre possible moviment.

Podríem definir aquests dos conceptes de la següent manera:

- **Perceptibilitat**: Fenomen produït quan una velocitat de càmera diferent de zero genera una velocitat 0 per alguns punts de la imatge. Un exemple de fenomen de perceptibilitat podria ser un moviment positiu en el eix z.

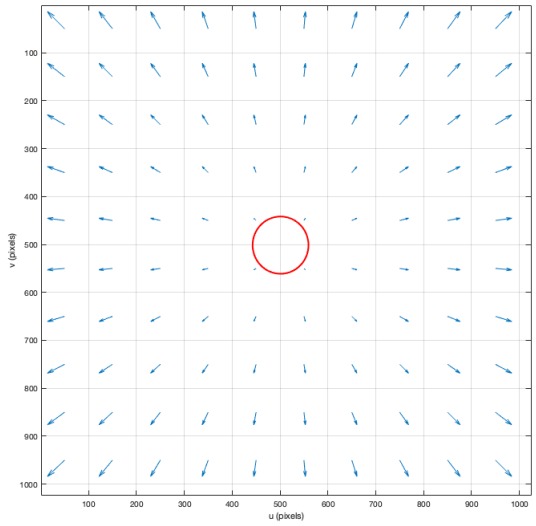


Figura 4.9: Flux òptic d'un moviment relatiu de translació *eix Z*

Com es pot comprovar en l'anterior figura, si ens fixem els punts centrals de la imatge tenen velocitat 0. Això implica que basant-nos en la informació que ens proporciona el *Jacobià* en aquest punt, no podem diferenciar estar quiet d'aquest moviment
Un altre exemple podria ser una rotació en z. Com es pot comprovar en la següent figura, els punts centrals també presenten velocitat 0.

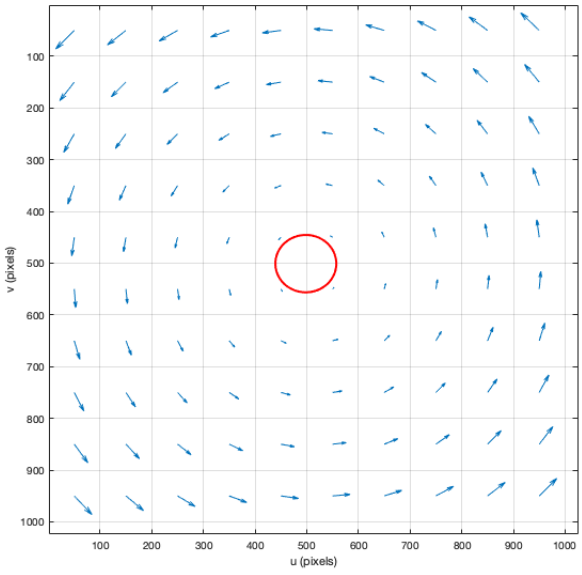


Figura 4.10: Flux òptic d'un moviment relatiu de rotació *eix Z*

- **Ambigüitat**: Fenomen generat quan dues velocitats de càmera diferent generen els mateixos canvis a certs punts de la imatge, és a dir que tenen la mateixa velocitat. Un exemple pràctic que es pot realitzar per comprovar aquest fenomen és moure el cap cap a la dreta

per veure com sembla que els objectes es mouen cap a l'esquerra. De la mateixa manera, si realitzem un moviment de rotació en sentit horari amb el cap també veurem com els elements es mouen cap a l'esquerra. Com més al centre del camp de visó, menys diferència.

Per tal de realitzar una anàlisi més acurada d'aquest fenomen, fem ús de l'aplicatiu *MATLAB* per presentar diferents situacions i veure la semblança en el flux òptic. De la mateixa manera que en el exemple gràfic, comparem una velocitat de translació en l'eix x amb una velocitat de rotació en el eix y.

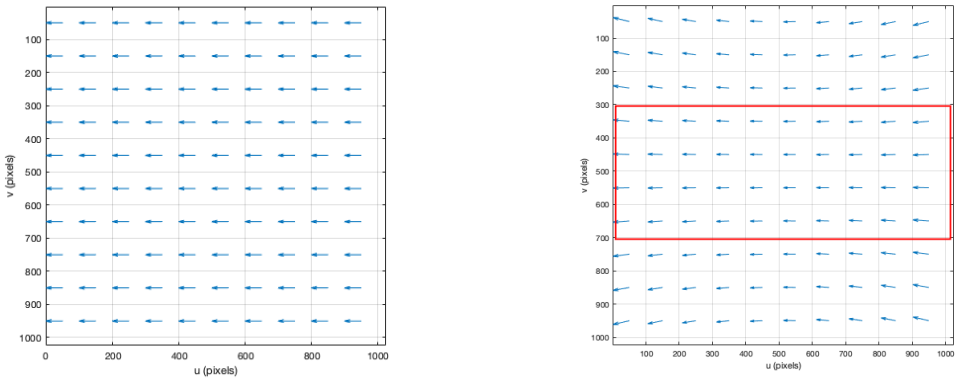


Figura 4.11: Flux òptic de traslació en eix x (esquerra) envers rotació en eix y (dreta)

Com es pot comprovar, la velocitat en el centre en la rotació en l'eix y és molt semblant a la produïda per una translació en l'eix x. A mesura que augmentem la distància focal, aquesta zona marcada en vermell en l'anterior figura augmenta. Per realitzar les anteriors imatges s'ha utilitzat una longitud focal de 8 mm. Si variem aquesta a una longitud de 30 mm el resultat per la rotació en y és el següent:

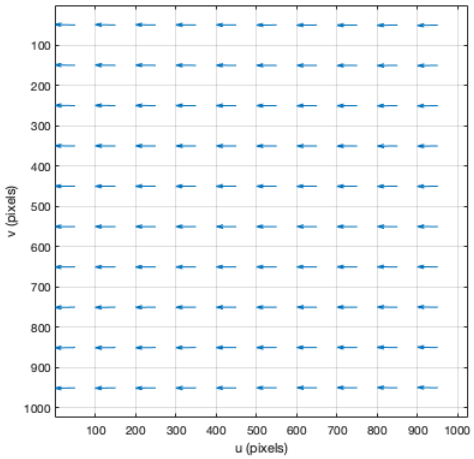


Figura 4.12: Flux òptic de translació en eix x (esquerra) envers rotació en eix y (dreta)

En canvi, si disminuïm la longitud focal, aquesta ambigüitat es fa més petita o pràcticament nul·la com es pot comprovar en la següent figura extreta d'un model amb una longitud focal

de 4 mm.

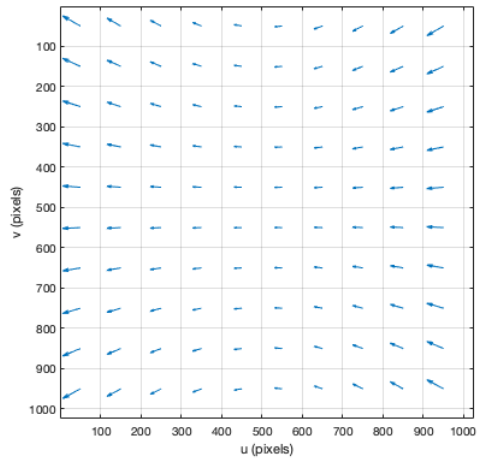


Figura 4.13: Flux òptic de translació en eix x (esquerra) envers rotació en eix y (dreta)

Podem explicar aquest fenomen si observem amb més detall el *Jacobià* de l'equació 14. Si ens fixem en els elements marcats en blau en la següent equació del *Jacobià*, són els termes que multipliquen a la velocitat x (v_x). De la mateixa manera, els marcats en vermell multipliquen a la velocitat de rotació y (w_y).

$$\begin{matrix} \frac{\lambda'}{Z} & 0 & \frac{-\bar{u}}{Z} & \frac{-\bar{u}\bar{v}}{\lambda'} & -\frac{\lambda'^2+\bar{u}^2}{\lambda'} & -\bar{v} \\ 0 & \frac{\lambda'}{Z} & \frac{-\bar{v}}{Z} & \frac{-\lambda'^2-\bar{v}^2}{\lambda'} & -\frac{\bar{u}\bar{v}}{\lambda'} & \bar{u} \end{matrix}$$

Si fem una anàlisi dels límits dels termes marcats en vermell per simular l'augment de la distància focal obtenim el següent:

$$\lim_{\lambda' \to \infty} \begin{pmatrix} -\frac{\lambda'^2+\bar{u}^2}{\lambda'} \\ -\frac{\bar{u}\bar{v}}{\lambda'} \end{pmatrix} = \begin{pmatrix} -\lambda' \\ 0 \end{pmatrix} \tag{15}$$

Observant els resultats obtinguts en l'anterior límit, veiem que és proporcional a la columna marcada en blau. De la mateixa manera, podríem fer una anàlisi semblant per les columnes marcades en marró per trobar un altre cas d'ambigüitat. No són els únics. Aquests errors d'ambigüitat es poden resoldre amb un sensor de rotació. Els humans solucionem aquest problema mitjançant uns receptors situats a l'orella que funcionen de forma semblant a un sensor de rotació.

4.4.1.3 Aplicacions del *Jacobià* Com s'ha comentat prèviament, el *Jacobià* ens aporta la informació de com varien les coordenades en el pla de la imatge respecte un moviment de la càmera en l'espai tridimensional. L'objectiu del present treball però és el contrari. Es pretén saber quina és la velocitat de la càmera necessària per tal d'aconseguir una posició desitjada dels paràmetres en el pla de la imatge.

Per plasmar aquest objectiu en forma d'equació, seguirem els següents passos on trobarem una relació entre la velocitat d'un punt detectat en el pla de la imatge i la velocitat de la càmera.

A partir de l'equació 14 i mitjançant la inversa del *Jacobià* podem trobar la següent equació:

$$\dot{\boldsymbol{r}} = \Big[\boldsymbol{J}(\boldsymbol{p1}, Z)\Big]^{+} \begin{bmatrix} \dot{u_1} \\ \dot{v_1} \end{bmatrix} \tag{16}$$

Degut a que només les matrius quadrades tenen inversa, utilitzem el recurs de la pseudoinversa de *Moore-Penrose* que es defineix com $A^{+} = (A^T A)^{-1} A^T$ L'objectiu ara és trobar la velocitat del punt exemple en el pla de la imatge. Per trobar-ho fem us d'un controlador lineal el qual ens dirigirà cap als punts desitjats en el pla d'imatge prèviament decidits. L'equació del controlador és la següent:

$$\dot{f} = \phi(f^* - f) \tag{17}$$

on f^* és la coordenada desitjada per el punt en el pla de imatge, f és la posició actual en píxels i ϕ és el controlador lineal. Ambdós coordenades estan referenciades al punt principal del pla de la imatge que podeu veure en la figura 4.7.

Finalment si combinem les equacions 16 i 17 trobem l'equació utilitzada per realitzar un control visual mitjançant informació extreta únicament de la imatge:

$$\boxed{\dot{\boldsymbol{r}} = \phi \Big[\boldsymbol{J}(\boldsymbol{p1}, Z)\Big]^{+} (f^* - f)} \tag{18}$$

Per trobar aquesta relació s'ha realitzat un exemple amb un punt extret del pla de la imatge. Es pot fer amb N punts però per un nombre de punts tal que $N \geqslant 3$ la matriu és molt difícil de condicionar si els punts són colineals o molt propers entre si com es pot veure a [2] ja que es poden generar problemes de perceptibilitat.

5 Tècniques de detecció

Per tal d'assolir l'objectiu del projecte és necessari detectar com a mínim una de les característiques objectiu dintre del pla de la imatge per fer-ne el seguiment. Aquesta detecció es realitza de diferents maneres dintre del gran ventall d'opcions que hi ha actualment en el camp de la visió per computador.

S'intenta no utilitzar tècniques molt complicades per així poder aprofundir en el coneixement d'aquestes. A més a més, és important no utilitzar algorismes amb un gran cost computacional, ja que perdríem continuïtat en el seguiment de l'objectiu.

Les metodologies escollides estan implementades dintre de la llibreria *OpenCV*. Una llibreria oberta pel camp de la visió per computador que implementa gran varietat de funcions molt útils en diferents llenguatges de programació com *Python* o *C++*.

Les metodologies escollides per desenvolupar el projecte són les següents:

- Color
- Marca visual *AprilTag*
- *Cascade Classifier*

A continuació s'expliquen en què consisteixen les metodologies escollides així com la implementació realitzada en *Python* per controlar el dron *DJI Tello*. A mesura que s'expliquen les diferents opcions també es presenta el resultat de la implementació.

5.1 Detecció del color

La primera metodologia escollida per la seva facilitat d'implementació i el baix cost de computació és la detecció per color.

L'objectiu d'aquesta detecció és ser capaç de detectar un color d'un objecte en la imatge per, mitjançant un petit algorisme, detectar el centre que més endavant s'utilitzarà com a entrada en l'algorisme de control.

5.1.1 Algorisme

L'algorisme utilitzat per tal de detectar el centre d'un objecte a partir del color es presenta en la següent figura. Aquest algorisme s'executa una vegada cada cop que llegim una imatge del dron. És a dir, si l'algorisme té un cost computacional alt pot derivar en un mal funcionament del seguiment.

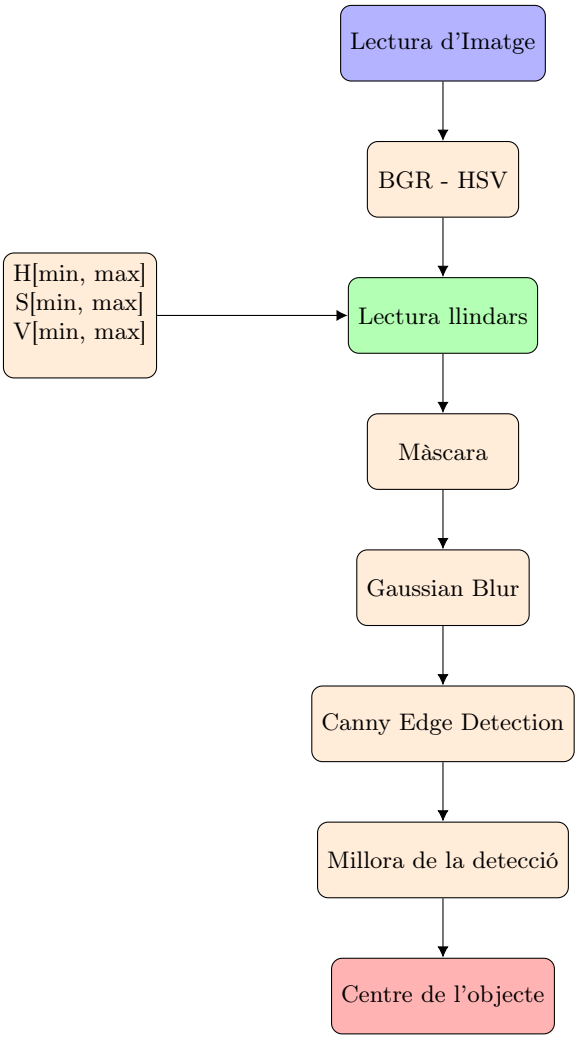


Figura 5.1: Algorisme detecció centre mitjançant color

El primer pas és detectar la imatge procedent del dron. Aquesta detecció la fem amb comandes enviades directament al dron mitjançant la connexió explicada en l'apartat 3.3 i la SDK del dron. A continuació transformem la imatge amb una funció implementada en *OpenCV*. La funció aplicada ens permet transformar la imatge rebuda, en format BGR (imatge en color formada pels plans blau, verd i vermell)), a format HSV. La raó principal d'aquesta transformació de l'espai de color és que el format HSV separa la intensitat de llum de la informació del color. Això és útil en moltes aplicacions en el camp de la visió per computador, ja que ens permet separar la informació que es manté constant (informació cromàtica) de la informació que pot variar en funció de les ombres o la intensitat de llum. Aquesta part d'informació se la coneix com *Luma*

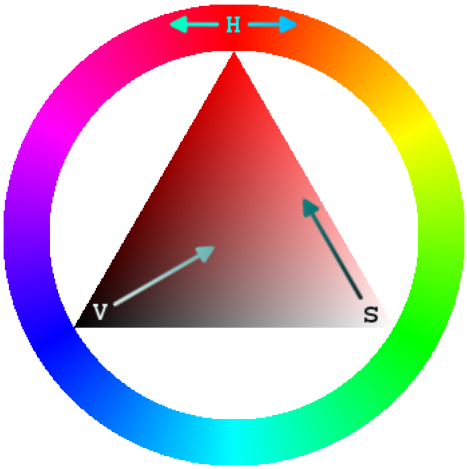


Figura 5.2: Espai de color HSV

Un cop tenim la imatge transformada a l'espai de color HSV, procedim a comparar els valors amb els llindars proporcionats per l'usuari per tal de detectar el color desitjat. Aquests valors conjuntament formen el color objectiu a seguir. S'introdueixen amb uns llindars per tal de poder acceptar petits canvis cromàtics sense perdre el control del dron. Aquests llindars són introduïts per l'usuari mitjançant uns *sliders* disponibles a la llibreria *OpenCV*.

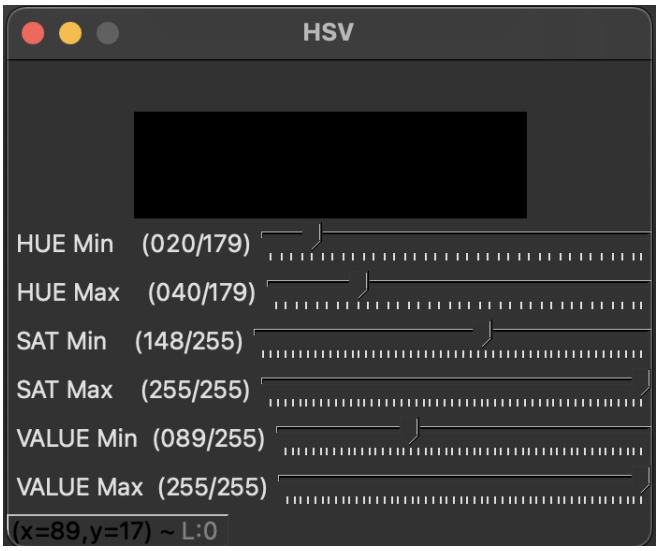


Figura 5.3: Control de paràmetres HSV per l'usuari

A continuació, també mitjançant una funció de *OpenCV* apliquem una màscara a tot el mapa de píxels. Els píxels que no estiguin dintre d'aquest llindars prèviament definits per l'usuari es definiran com color negre, com podeu veure a la figura 5.11. L'objectiu és separar el color d'interès de la resta de la imatge per així poder fer els següents passos amb menys informació i en conseqüència major eficiència.

Un cop tenim implementada la màscara, només treballem amb la imatge en la regió d'interès. Procedim a realitzar un filtre anomenat *Gaussian Blur*. Aquest filtre consisteix en aplicar un *ker-*

nel a cada agrupació de píxels per extreure un únic valor representatiu. Es pot veure gràficament a la següent figura:

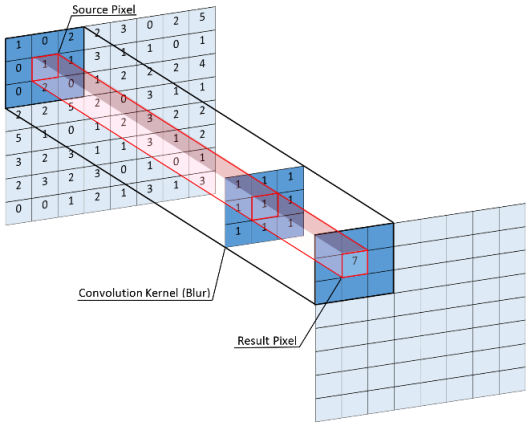


Figura 5.4: Procediment gràfic *kernel*

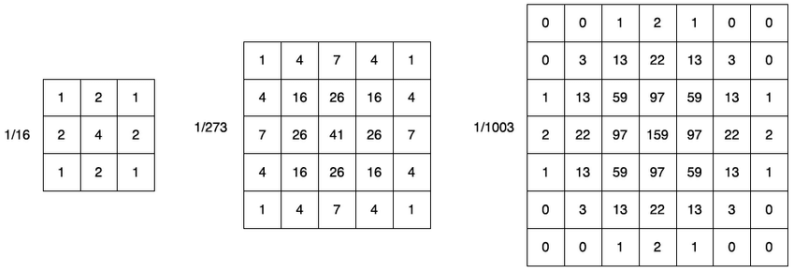


Figura 5.5: Exemple *kernels* per *Gaussian Blurr*: 3x3, 5x5, 7x7

L'objectiu és separar el fons de la imatge de l'objecte en si per posteriorment detectar el contorn de l'objecte. Això és a causa del fet que si un objecte està més a prop que la resta d'objectes que conformen la imatge, també tindrà més píxels que el defineixen i el filtratge mitjançant la funció *Gaussian Blur* afectarà menys a aquest, com es pot veure en la següent figura:

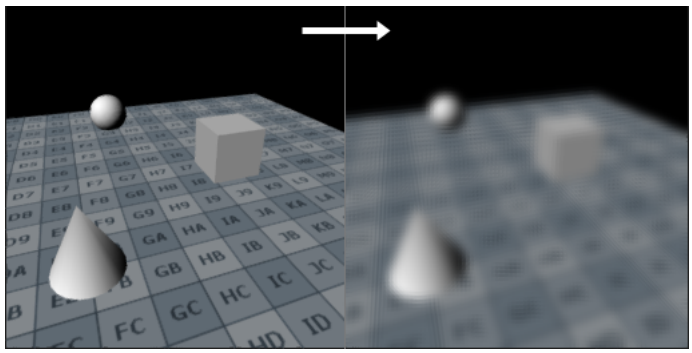


Figura 5.6: Exemple d'aplicació algorisme *Gaussian Blur*

A més a més, aquest procediment també permet homogeneïtzar la imatge i optimitzar la detecció, ja que l'algorisme de detecció de contorns *Canny* serà capaç de detectar els contorns exteriors i

evitarem agafar els interiors. Com es pot veure en la següent figura, quan procedim a detectar els contorns presents en una imatge sense prèviament aplicar un filtre *Gaussian Blur* la detecció es fa molt més complexa i més difícil de tractar. Realment el filtre *Gaussian Blur* elimina el soroll d'alta freqüència de la imatge, cosa que simplifica la detecció del contorn.

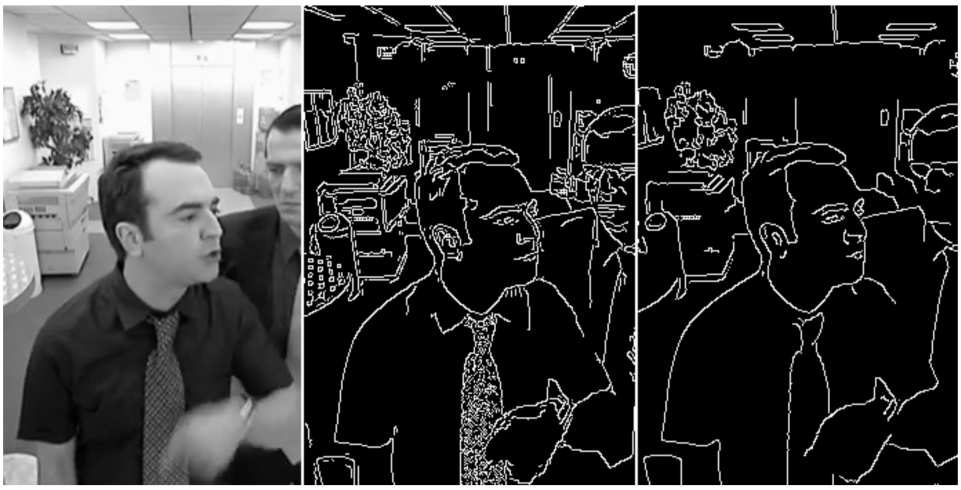


Figura 5.7: Detecció de contorns amb (dreta) i sense (centre) *Gaussian Blur* mitjançant *Canny*

Un cop hem realitzat el filtre, procedim a implementar l'algorisme d'extracció de contorns de *Canny* amb la funció *CannyDetect* amb l'objectiu de detectar el contorn del nostre objecte per a continuació, mitjançant una interpolació matemàtica trobar el centre de l'objecte. El primer pas teòric per a la realització d'aquest algorisme és la reducció de soroll present a la imatge. A continuació l'algorisme aplica una convolució amb un *kernel* de *Sobel* en la direcció x i y del pla d'imatge. Amb aquest pas i mitjançant dues simples fórmules aconseguim el gradient de contorn i direcció per a cada píxel. Després d'obtenir el gradient i la seva direcció per a cada píxel, es processa un altre cop la imatge per complet per eliminar els píxels que poden no constituir el contorn.
Per veure-ho gràficament, en la següent figura el punt A es troba en un contorn. Els punts B i C es troben en la direcció del gradient. A continuació es comprova si el punt A és un màxim local comparat amb el punt B i C. Si ho és, avança al següent pas. En cas contrari es suprimeix, és a dir se li dona valor 0.

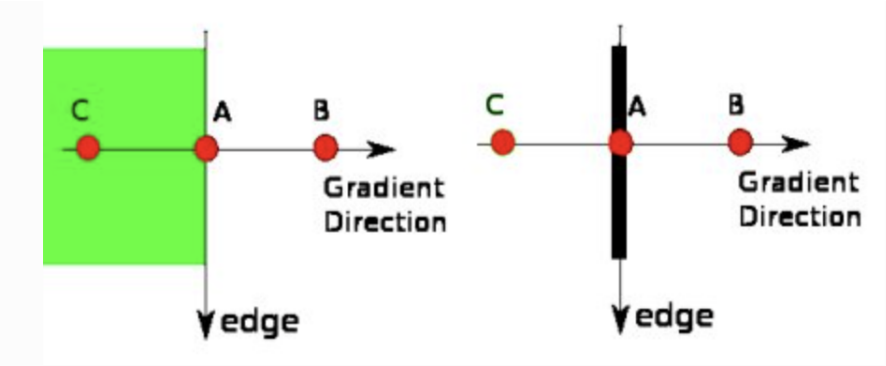


Figura 5.8: Exemple gràfic supressió dels valors no màxims *(non-maxima supression)*

L'últim pas d'aquest algorisme consisteix en realitzar un filtrat als punts que en el pas anterior s'han declarat com "possibles contorns". Per a la realització d'aquest últim pas necessitem dos valors llindars. El valor mínim i el valor màxim. Aquells contorns amb una intensitat major al valor màxim seran considerats com a contorns segurs. Aquells per sota del valor mínim seran directament descartats. Per últim, aquells que estiguin compresos entre el valor mínim i màxim tindran un tractament especial, si estan connectats a un píxel catalogat com contorn segur, també ho serà considerat. En cas contrari serà descartat.

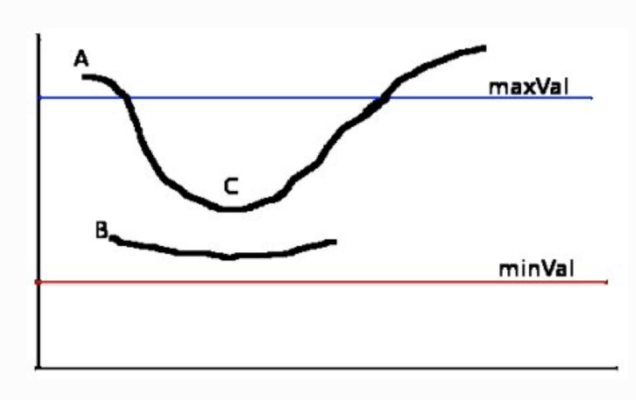


Figura 5.9: Limits de detecció per algoirsme *Canny*

Abans de procedir a calcular el centre de l'objecte en funció del contorn, estudiem un seguit d'operacions que ens permeten millorar la detecció d'aquests. A aquest conjunt d'operacions se les coneix com a transformacions morfològiques i entre d'altres hi ha:

- Erosió
- *Opening*
- Gradient
- Diltació
- *Closing*
- *Top Hat*

(a) Original (b) Erosió (c) Dilatació (d) *Opening*

(e) *Closing* (f) *Gradient* (g) *Top Hat*

Figura 5.10: Transformacions morfologiques habituals disponibles a la llibreria *OpenCV*

En el nostres cas, després de diferents proves s'ha realitzat una dilatació del contorn generat per

l'algorisme *Canny*. Tot seguit, s'executa una funció disponible a la llibreria *OpenCV* per tal de detectar els contorns tancats que poden haver-hi a la imatge.

Per últim, un cop tenim el contorn de l'objecte detectat, passem a calcular el centre el qual després s'utilitzarà com a entrada del programa de control. Per tal de detectar el centre, hem de poder representar de manera matemàtica el contorn tancat prèviament detectat. Per això utilitzem tres funcions que ens permeten aproximar l'objecte a un rectangle. Un cop tenim les coordenades de les quatre cantonades, aproximem el centre (c_x, c_y) mitjançant la següent fórmula:

$$\begin{aligned} c_x &= x + \frac{w}{2} \\ c_y &= y + \frac{h}{2} \end{aligned} \tag{19}$$

On w i h son l'ample i alt del rectangle envolvent.

Hi han diferents formes de calcular el centre de l'objecte en funció de la precisió desitjada. En aquest cas s'ha aproximat a un quadrat mitjançant les funcions explicades, però es podria haver tingut en compte tots els píxels de l'objecte i aplicar la funció *cv2.moments* que realitza una aproximació semblant al centre de gravetat però amb píxels.

En les següents figures es poden veure els resultats de la implementació:

Figura 5.11: Implementació en DJI Tello. Màscara

Un cop tenim la zona de color detectada, procedim a aproximar-la amb un polinomi per posteriorment obtenir el centre.

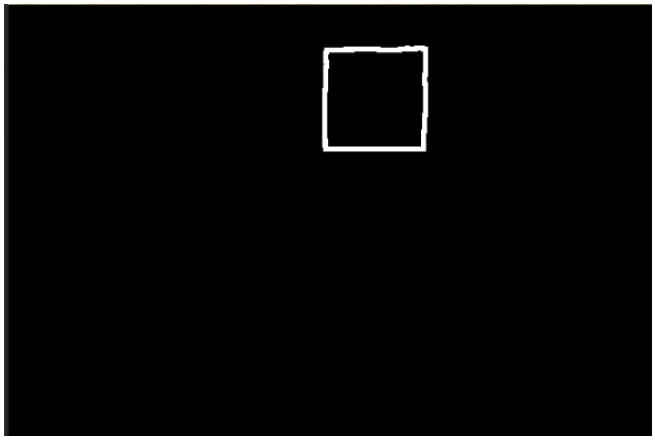


Figura 5.12: Aproximació rectangle detecció

A continuació es mostra el codi implementat en *Python*:

```python
#Creacio funcio per detectar contorn i calcular centre
def getContours(img, imgContour):
    global dir
    #Deteccio contorns:
    contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_NONE)
    if len(contours) != 0:
      #Ens quedem amb el major contorn
        cnt = max(contours, key=cv2.contourArea)
        area = cv2.contourArea(cnt)
        areaMin = cv2.getTrackbarPos("Area", "Parameters")
        #Discretitzem si l'area trobada es suficientment gran
        if area > areaMin:
            cv2.drawContours(imgContour, cnt, -1, (255, 0, 255), 7)
            #Aproximem el contorn a un rectangle
            peri = cv2.arcLength(cnt, True)
            approx = cv2.approxPolyDP(cnt, 0.02 * peri, True)
            x, y, w, h = cv2.boundingRect(approx)
            #Calculem el centre
            cx = int(x + (w / 2))  # CENTER X OF THE OBJECT
            cy = int(y + (h / 2))  # CENTER X OF THE OBJECT

```

Codi 5.1: Funció per detectar contorn i calcular centre

```python
while True:
    #Obtenim imatge del dron Tello
    frame_read = drone.get_frame_read()
    droneFrame = frame_read.frame
    img = cv2.resize(droneFrame, (width, height))
    #Transformem el espai de color de BGR a HSV
    imgHsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    #Guardem en variables els valors seleccionats per l'usuari per determinal el
     color
    h_min = cv2.getTrackbarPos("HUE Min", "HSV")
    h_max = cv2.getTrackbarPos("HUE Max", "HSV")
    s_min = cv2.getTrackbarPos("SAT Min", "HSV")
    s_max = cv2.getTrackbarPos("SAT Max", "HSV")
    v_min = cv2.getTrackbarPos("VALUE Min", "HSV")
    v_max = cv2.getTrackbarPos("VALUE Max", "HSV")
    #Apliquem la mascara en funcio del color
    lower = np.array([h_min, s_min, v_min])
    upper = np.array([h_max, s_max, v_max])
    mask = cv2.inRange(imgHsv, lower, upper)
    result = cv2.bitwise_and(img, img, mask=mask)
    mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)
    #Tractament de la imatge per detectar el contorn
    imgBlur = cv2.GaussianBlur(result, (7, 7), 1)
    imgGray = cv2.cvtColor(imgBlur, cv2.COLOR_BGR2GRAY)
    limit1 = cv2.getTrackbarPos("Threshold1", "Parameters")
    limit2 = cv2.getTrackbarPos("Threshold2", "Parameters")
    imgCanny = cv2.Canny(imgGray, limit1, limit2)
    # Millora de la deteccio de contorn
    kernel = np.ones((5, 5))
    imgDil = cv2.dilate(imgCanny, kernel, iterations=1)
    getContours(imgDil, Contour)
    display(imgContour)

```

Codi 5.2: Programa principal detecció mitjançant color

5.2 AprilTag

El següent mètode per a la detecció d'objectius escollit per la realització d'aquest projecte és mitjançant la tecnologia de marques visuals de referència. Les marques visuals són referències artificials dissenyades específicament per ser reconegudes fàcilment. Un altre sistema de barres en dues dimensions, molt utilitzada actualment, i que ens permet fer analogia és el sistema QR. A diferència d'aquests sistemes, les marques visuals disposen d'una càrrega d'informació molt més petita. Normalment fins a 12 bits. El que les fa molt més eficients per al nostre objectiu i per al món de la robòtica en general.

L'objectiu és utilitzar les marques visuals com a referència geomètrica a la imatge. Coneixent les dimensions i la composició d'aquesta marca visual en conjunt amb un seguit de transformacions matemàtiques simples som capaços d'extrapolar aquesta informació i detectar aquesta marca visual especifica a la imatge.

En la següent figura extreta de [4] es pot veure un seguit de marques visuals diferents:

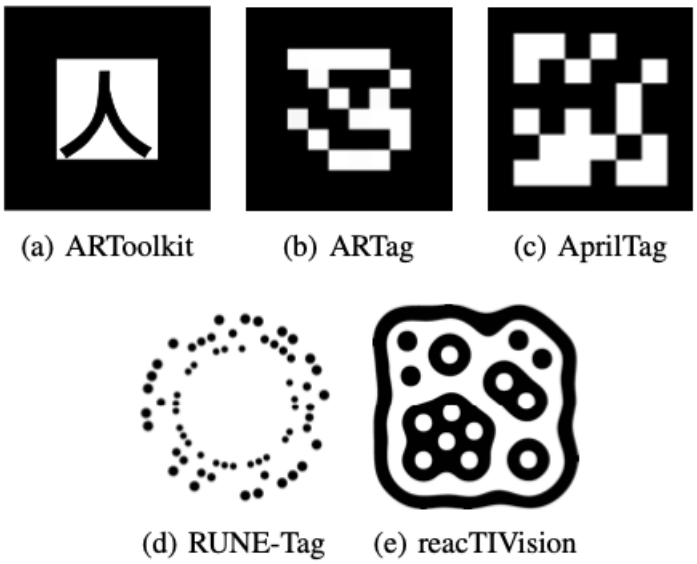


Figura 5.13: Exemple de marques visuals més utilitzades a l'indústria

Per aquest projecte en concret s'ha decidit utilitzar les marques visuals *AprilTag* per diverses raons. La primera és la seva fàcil implementació amb *Python*, ja que s'ha dissenyat un mòdul amb algunes funcions. La segona, el baix cost computacional. Grans empreses com la NASA utilitzen aquestes marques visuals en els seus projectes. *AprilTag* és un sistema de detecció visual basat en marques visuals útil per gran varietat de funcions com per exemple realitat augmentada, robòtica o calibratge de càmera entre d'altres. Aquests *AprilTag* poden ser fàcilment creats amb una impressora i el software de detecció és capaç de precisar la posició en 3 dimensions i fins i tot l'orientació. Mitjançant aquests sistemes es pot aconseguir una detecció a temps real degut al seu baix cost computacional. Està específicament dissenyat per ser integrat en altres aplicacions de manera molt senzilla al mateix temps que portable. En la següent taula podem veure diferents aspectes a favor i en contra dels sistemes *AprilTag*.

Avantatges	Inconvenients
Llicència lliure	Només implementat per ROS
Pocs paràmetres de configuració	Falses deteccions
Treballa bé a llargues distàncies	
Utilitzat per la NASA	
Disseny flexible (no sempre quadrat)	
Baix cost computacional	
Personalitzable	
Mòdul amb algunes funcions	

Taula 5.1: Pros i contres de la tecnologia *AprilTag*

A continuació s'expliquen els diferents passos duts a terme pel sistema de detecció per identificar un *AprilTag* en la imatge. De manera simplificada podem dir que el detector intenta buscar regions quadrades que tenen un interior més fosc que l'exterior. Podeu trobar més informació a [5].

Metodologia *AprilTag*

1. Detecció de línia
   S'aplica el gradient de direcció a cada píxel amb l'objectiu de permetre la detecció de segments de línia en la imatge. Utilitza un mètode molt eficient descrit a [6].

2. Detecció de quadrat
   Tot seguit, l'algorisme computa una primera cerca entre tots els segments. Si quatre línies intersecten de manera adequada per formar un quadrat, aquesta regió es seleccionada com a candidata de detecció.

3. Homografia i estimació extrínseca
   Un cop el quadrat es detectat l'homografia es calculada mitjançant un algorisme anomenat *Direct Linear Transform (DLT)*. Després mitjançant un altre algorisme és capaç de trobar el vector rotació i el vector translació d'aquella regió quadrada.

4. Comprovació de carga
   Un cop tenim el vector translació i el vector rotació, el mètode calcula quina és la càrrega del quadrat. Usant la homografia abans calculada és capaç d'identificar si és un *tag* conegut o no.

Existeixen diferents categories dintre del paquet *AprilTag*. Aquestes categories constitueixen maneres d'agrupar els bits d'informació en diferents formes en funció de la geometria on es col·loquen aquestes marques visuals. En la següent figura podem veure les sis famílies diferents:

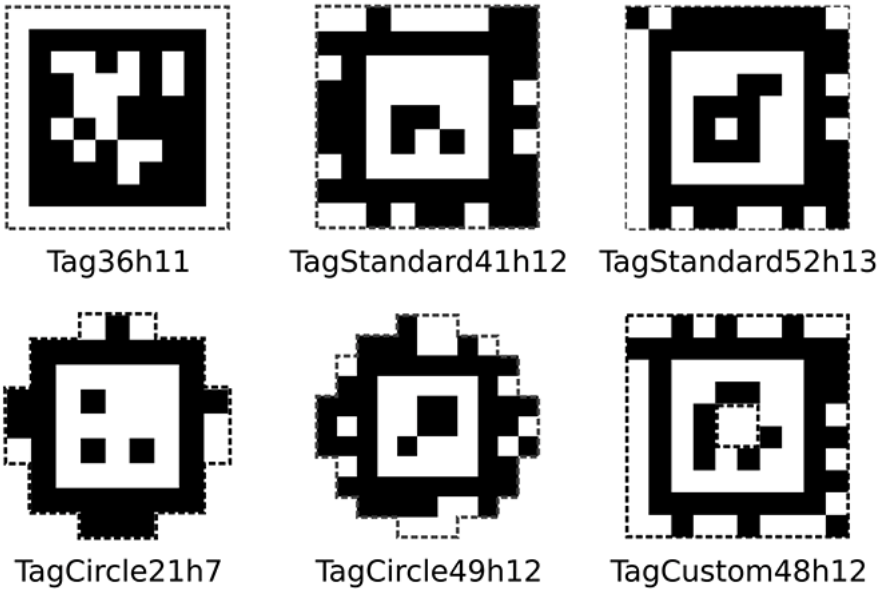


Figura 5.14: Exemple de les sis families de *AprilTag* diferents

A més a més, dintre de cada família es poden realitzar diferents agrupacions. Aquestes agrupacions tenen un ID característic calculat a partir del patró intern de la marca visual. Aquest nivell de personalització causa una gran varietat de combinacions permetent així que podem disposar d'un identificador quasi únic.

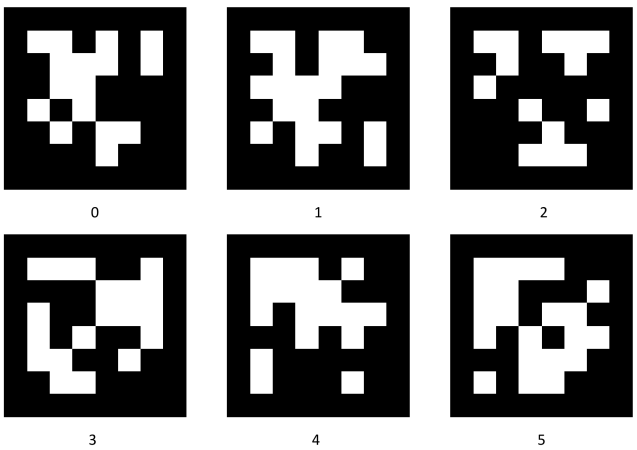


Figura 5.15: Exemple de marques visuals de la família Tag36h11 amb un ID únic.

A continuació es mostra el resultat d'aplicació d'aquest algorisme en una imatge extreta de la pàgina oficial d'aquest software. Per afegiment i com hem comentat prèviament, gràcies al fet que la composició d'aquestes marques visuals és coneguda podem identificar fàcilment l'orientació.

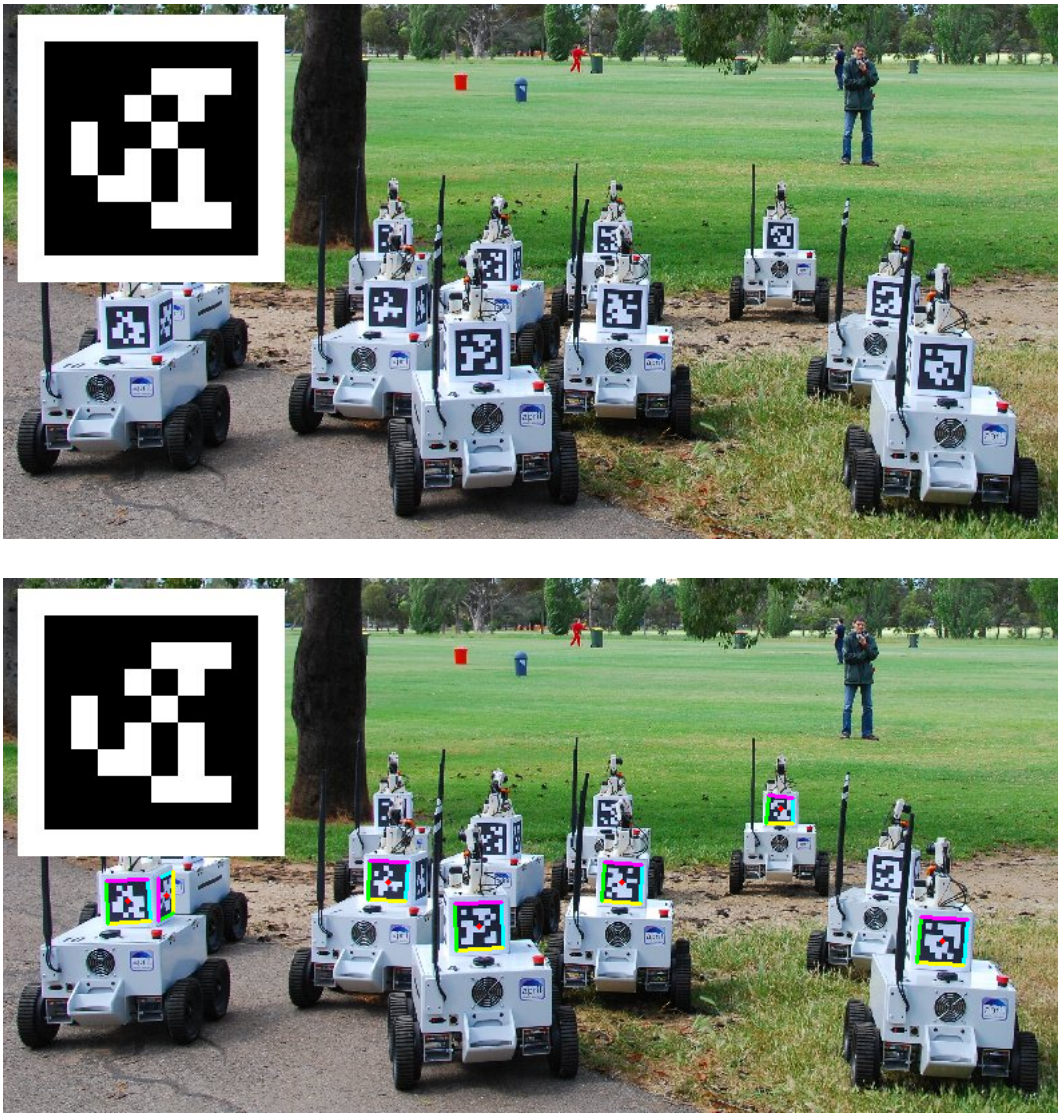


Figura 5.16: Resultat d'aplicació detecció *AprilTag*

En l'anterior imatge es pot comprovar la capacitat de detecció en funció de la rotació dels *AprilTags* respecte a la perpendicular de la imatge. També es pot comprovar que si un objecte cobreix parcialment el *tag* es perd la detecció. Podeu trobar més informació sobre les limitacions de la tecnologia a [7].

5.2.1 Algorisme

L'algorisme utilitzat té el mateix objectiu que l'anterior comentat. Detectar el centre de l'objecte, en aquest cas del *AprilTag* per així després, mitjançant el mètode de control adequat, poder realitzar el control del dron. De la mateixa manera que l'anterior, aquest algorisme s'executa un cop cada vegada que llegim una imatge que prové del dron.

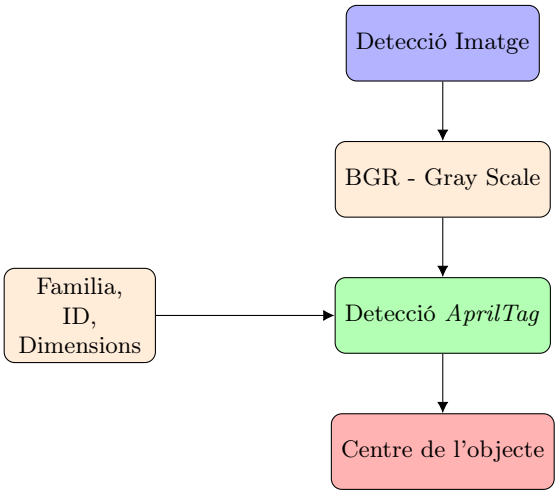


Figura 5.17: Algorisme detecció centre mitjançant *AprilTag*

El centre del *AprilTag* és calculat automàticament pel mètode. Un cop s'ha realitzat la detecció només cal cridar la funció que calcula el centre a partir d'una comanda amb *Python*.

A continuació es mostra el codi implementat en *Python*:

```
#Definicio de funcio
def getAprilTag(image, gray, options):
    #Deteccio del apriltag en funcio de les opcions especificades
    detector = apriltag.Detector(options)
    results = detector.detect(gray)
    #Comprovem si tenim deteccio
    if len(results) != 0:
      #Tractament dels resultats obtinguts per la funcio "detector"
        r = results[0]
        (ptA, ptB, ptC, ptD) = r.corners
        ptA = (int(ptA[0]), int(ptA[1]))
        ptB = (int(ptB[0]), int(ptB[1]))
        ptC = (int(ptC[0]), int(ptC[1]))
        ptD = (int(ptD[0]), int(ptD[1]))
        #Remarquem el apriltag en la imatge
        cv2.line(image, ptA, ptB, (0, 255, 0), 2)
        cv2.line(image, ptB, ptC, (0, 255, 0), 2)
        cv2.line(image, ptC, ptD, (0, 255, 0), 2)
        cv2.line(image, ptD, ptA, (0, 255, 0), 2)
        #Obtenim el centre del apritag
        (cx, cy) = (int(r.center[0]), int(r.center[1]))
        cv2.circle(image, (cx, cy), 5, (0, 0, 255), -1)
        #Calculem el area del AprilTag
        area = 0.5 * ((ptA[0]*ptB[1]-ptA[1]*ptB[0]) +
                      (ptB[0]*ptC[1]-ptB[1]*ptC[0]) +
                      (ptC[0]*ptD[1]-ptC[1]*ptD[0]) +
                      (ptD[0]*ptA[1]-ptD[1]*ptA[0]))
```

Codi 5.3: Funció per detectar *AprilTags*

```
#Programa principal
while True:
    #Llegim la imatge del dron
    frame_read = drone.get_frame_read()
```

```
    myFrame = frame_read.frame
    img = cv2.resize(myFrame, (width, height))
    # Transformem del espai de colors BGR a escala de grisos
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Especificquem les opcions de deteccio
    options = apriltag.DetectorOptions(families='tag36h11')
    # Crida a la funcio creada previament
    getAprilTag(img, gray, options)
    display(img)
```

Codi 5.4: Programa principal detecció mitjançant *AprilTags*

5.3 Detecció facial amb *Cascade Classifier*

L'última metodologia implementada és la detecció facial d'una persona a través d'un classificador en cascada a partir de les anomenades màscares de Haar.
El classificadors en cascada basats en funcions *Haar* és un mètode eficaç de detecció proposat per Paul Viola i Michael Jones en el seu document *"Rapid object detection using a Boosted Cascade of simple features"* al 2001 [8]. Es tracta d'un punt de vista basat en l'aprenentatge automàtic en el qual la funció en cascada s'entrena a partir de moltes imatges positives i negatives. És a dir, que contenen o no l'objecte a detectar. En el nostre cas l'objecte a detectar és el rostre d'una persona.

Per extreure les característiques de la imatge s'utilitzen les característiques de Haar que podeu veure a continuació:

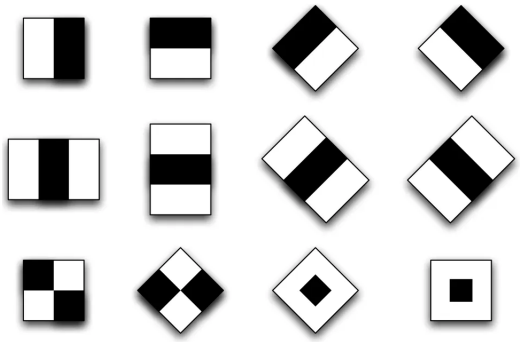


Figura 5.18: Principal funcions Haar

Cada tipus de funció obté un valor individual calculat a partir de la resta entre la suma de píxels sota el rectangle blanc amb el nombre de píxels sota el rectangle negre.

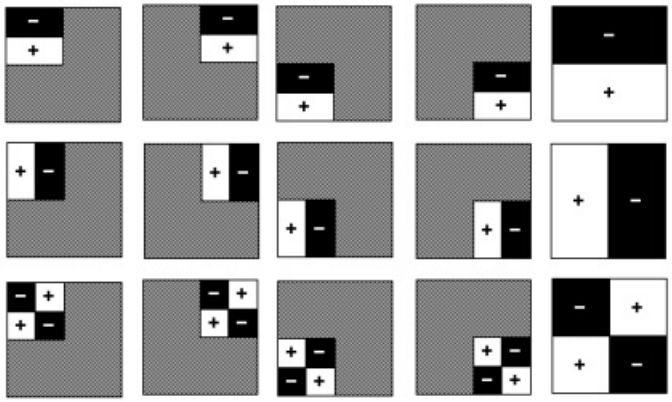


Figura 5.19: Caracteristiques de Haar sota rotacions, translacions i canvis d'escala

En el treball presentat per Viola-Jones les característiques a l'inici es calculen mitjançant *kernels* de 24x24 en imatges en blanc i negre. Semblants a les màscares utilitzades en l'apartat 5.1. Aquests càlculs donen lloc a un gran nombre de combinacions que fan molt difícil l'eficiència d'aquest sistema. Com a conseqüència, Viola-Jones van introduir a l'any 2001 el concepte intermedi de **Imatge Integral** [8].

Aquest càlcul intermedi es realitza mitjançant la següent fórmula:

$$ii(x,y) = \sum_{x'<=x,y'<=y} i(x',y'), \tag{20}$$

on $ii(x,y)$ és la imatge integral i $i(x,y)$ la imatge original. Per entendre aquesta idea, ens ajudem de la següent figura:

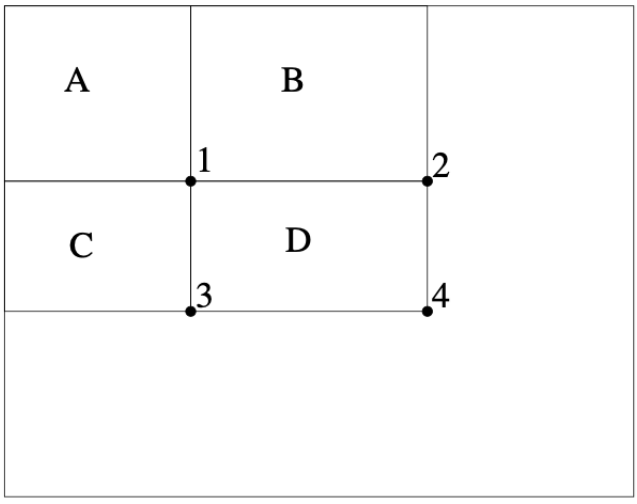


Figura 5.20: Concepte d'imatge integral

El valor a la localització 1 correspon a la suma de tots els píxels corresponents a la zona A. La localització 2 correspon a la suma dels rectangles A i B. El punt 3 correspon a la suma de A i C

mentre que el punt 4 és la suma de tots (A,B,C,D). Tenim:

$$\begin{aligned} 1 &\equiv A \\ 2 &\equiv A + B \\ 3 &\equiv A + C \\ 4 &\equiv A + B + C + D \end{aligned}$$

Fàcilment podem veure que per calcular, per exemple l'àrea B, podem fer $2 - 1$. De la mateixa manera, si volem calcular l'àrea de la regió D ho podem fer agregant els valors dels punts anteriors de la següent manera: $4 + 1 - (2 + 3)$. Matemàticament això es pot expressar amb la següent recurrència:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{21}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{22}$$

On $s(x, y)$ es la suma acumulada per fila.

Aquest pas intermedi permet augmentar l'eficiència d'aquest mètode a l'hora de ser implementat en un algorisme real, ja que permet calcular la suma dels píxels sota una regió amb un càlcul que involucra només quatre punts.

Com s'ha comentat al principi de la secció, l'algorisme és entrenat amb grans quantitats d'imatges positives i negatives per tal de detectar quines són les màscares, orientació i posició que conjuntament formen, en aquest cas, un rostre facial. Aquest treball és realitzat mitjançant tècniques de *Machine Learning* i amb grans *sets* de dades. Per entendre gràficament a què ens referim amb la detecció de característiques en les imatges mitjançant les màscares de Haar, tenim la següent imatge:

Figura 5.21: Barack Obama, 44è President dels Estats Units d'Amèrica

Com es pot comprovar en la imatge, si en fixem, podem veure que per exemple els píxels en els llavis són més foscos que en la part superior d'aquests. També podem veure com als laterals del nas els píxels també tenen una intensitat superior respecte al centre del nas. Mitjançant les màscares de Haar i el concepte introduït anteriorment de la imatge integral, podem calcular

quines zones de la imatge poden contenir una característica típica del rostre humà. Com també hem comentat, aquest algorisme ha estat prèviament entrenat amb grans *sets* de dades permetent així conèixer quines són les màscares i dimensions més eficients per a detectar-lo. En la següent figura es poden veure diferents màscares de Haar que juntes són capaces d'identificar una cara.

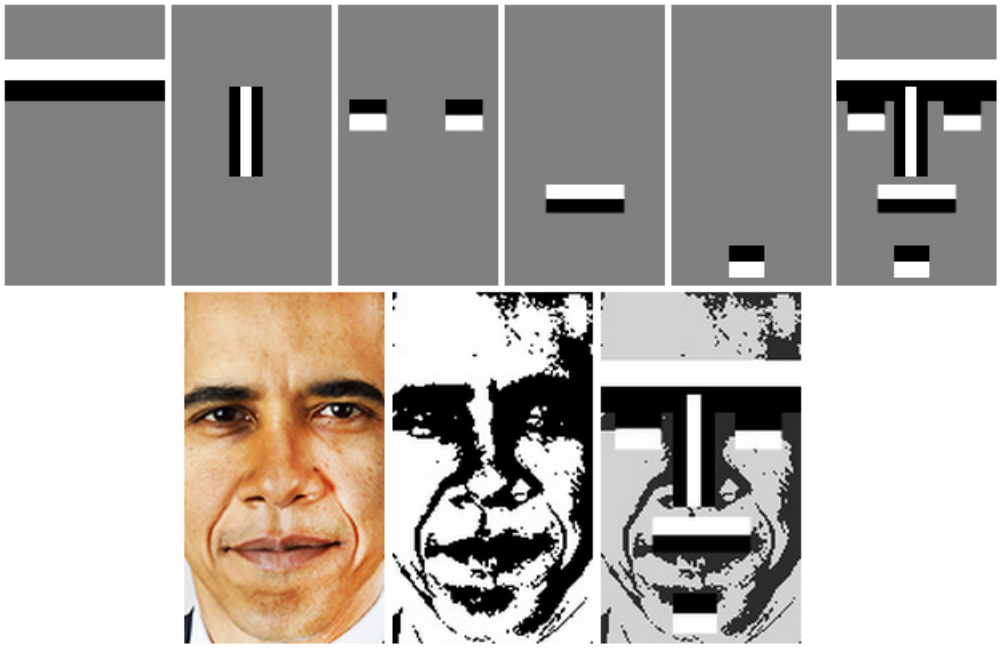


Figura 5.22: Detecció de característiques mitjançant màscares de Haar

L'algorisme és capaç de detectar si troba una possible característica de Haar, ja que al realitzar la resta entre la suma de la intensitat dels píxels sota el rectangle negre i la suma de la intensitat dels píxels sota el rectangle blanc ha de ser molt propera a 1.

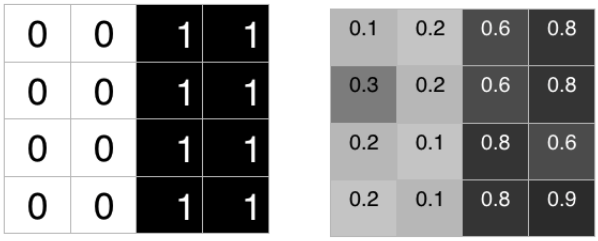


Figura 5.23: Característica Haar ideal (esquerra) envers real (dreta)

Mitjançant la següent equació comprovem com s'assembla la característica real a la ideal. Un resultat de 1 equival a exactament igual i 0 totalment diferent:

$$\Delta = fosc - clar = \frac{1}{n}\sum_{dark} i(x) - \frac{1}{n}\sum_{white} i(x) \tag{23}$$

En funció del llindar de precisió que definim, el resultat de l'anterior operació ens dirà si l'element detectat és una característica o no. Un cop ha detectat totes les característiques de la imatge,

l'algorisme realitza una combinació lineal prèviament entrenada mitjançant l'algorisme *AdaBoost* per determinar si la combinació de les característiques detectades formen una cara o no.
Un altre problema sorgeix quan el resultat de l'entrenament amb 4916 cares diferents i 75 milions de finestres resulta en 6061 característiques diferents identificades mitjancant màscares de Haar. Aplicar els càlculs esmentats anteriorment 6061 vegades sense saber quina dimensió tindrà la cara ni quina resolució té el dispositiu és molt poc eficient computacionalment parlant. Per solucionar aquest problema, Viola-Jones van donar amb el concepte de **Classificador en cascada**.

El classificador en cascada, descrit en [8] consisteix en dividir les 6061 màscares característiques de Haar que identifiquen un rostre en diferents etapes, com si fos un arbre de desició. En cas que una regió no passes la primera etapa, seria descartada completament. Permetent així eliminar gran part de la regió de la imatge de manera molt ràpida i eficaç per poder centrar la potència de càlcul en aquelles regions que si són susceptibles de tenir una cara. En el següent flux podeu veure aquesta idea:

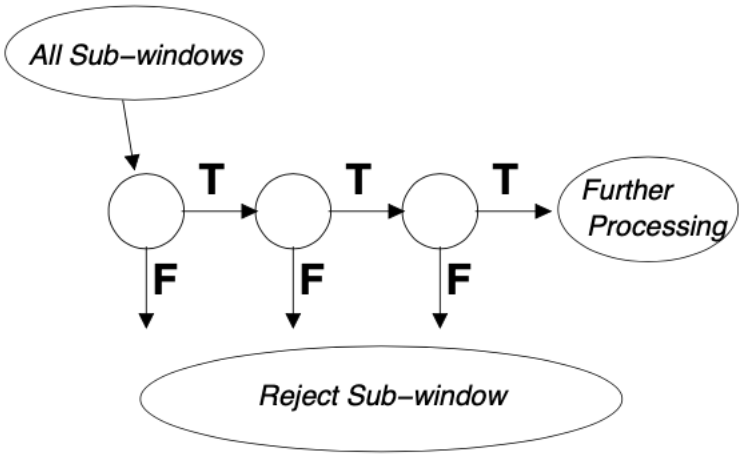


Figura 5.24: Diagrama de flux classificació en cascada.

En el document escrit per Viola-Jones [8] aquestes etapes són anomenades *stages*. En total disposen de 38 etapes diferents. El nombre de característiques de Haar en cada etapa és creixent segons: 1, 10, 25, 25, 50..
Durant l'entrenament d'aquest classificador en cascada mitjançant el meta algorisme *Ada Boost* es decideix quines són les característiques de Haar que s'apliquen i en quin ordre per fer el procés més eficient. En concret, les dues primeres característiques utilitzades per discriminar són les dels ulls i el nas.

Degut a l'augment i millora de tècniques de *Deep Learning* existeixen moltes altres tècniques per detectar i/o identificar tot tipus d'objectes però el cost computacional és molt superior. En aquest projecte s'ha prioritzat tenir un cost computacional baix per poder tenir un correcte funcionament amb el dron. Cal remarcar que tot i que s'han trobat altres formes més robustes de detecció, la detecció mitjançant classificadors en cascada de Haar és una tècnica àmpliament utilitzada avui en dia degut al seu baix cost computacional. Qualsevol petit dispositiu que és capaç de detectar rostres és molt probable que estigui utilitzant aquesta tècnica. Com per exemple, la detecció facial de la càmera dels nostres dispositius *smartphone*.

Per últim comentar que a l'actualitat existeixen grans quantitats de classificadors en cascada que poden detectar tot tipus d'objectes i/o formes degut a la facilitat d'entrenament d'aquests sistemes i el seu baix cost computacional, com podeu veure a [9]. A continuació es llisten els classificadors en cascada més coneguts:

- Cara frontal
- Cara lateral
- Tronc superior
- Tronc inferior
- Tronc sencer
- Ulls
- Somriure
- Matrícula de vehicle

5.3.1 Algorisme

L'algorisme implementat és molt semblant als dos algorismes explicats anteriorment. L'objectiu és el mateix: Obtenir el centre que tot seguit servirà com a entrada al sistema de control. En aquest cas, es seguiran els passos més adients per tal de facilitar la detecció al ***Cascade Classifier***. Un cop tinguem la detecció, procedirem a aproximar la regió en un quadrat, per així poder calcular el centre amb gran facilitat mitjançant l'equació 19. Es poden veure els passos en el següent diagrama:

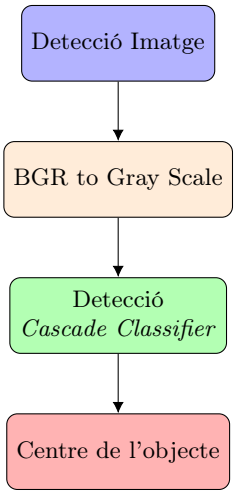


Figura 5.25: Algorisme detecció centre mitjançant *Cascade Classifier*

A continuació es mostra el codi implementat en *Python*:

```
#Definicio de la funcio
def getContours(img):
     #Transformem de l'espai de color BGR a escala de grisos
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #Utilitzem el classificador en cascada per detectar "frontal face"
    face = face_cascade.detectMultiScale(gray, 1.3, 5)
    maxArea = []
    #Comprovem si hi ha alguna deteccio
    if len(face) != 0:
        for (x, y, w, h) in upper:
      #Obtenim 4 punts a partir del classificador
            ptA = [x+w, y+h]
            ptB = [x+w, y]
            ptC = [x, y]
            ptD = [x, y+h]
            #Calculem l'area
            area = 0.5 * ((ptA[0]*ptB[1]-ptA[1]*ptB[0]) +
                          (ptB[0]*ptC[1]-ptB[1]*ptC[0]) +
                          (ptC[0]*ptD[1]-ptC[1]*ptD[0]) +
                          (ptD[0]*ptA[1]-ptD[1]*ptA[0]))
            #Ens quedem amb el rostre mes proper
            if area >= maxArea:
                maxArea = area
                x_ = x
                y_ = y
                w_ = w
                h_ = h
  #Calculem el centre
        cv2.rectangle(img, (x_, y_), (x_+w_, y_+h_), (0, 255, 255), 2)
        cx = (x_+w_)/2
        cy = (y_+h_)/2
```

Codi 5.5: Funció per detectar cara

```
#Programa principal

#Cridem al classificador en cascada i carguem el arxiu xml pertinent
face_cascade = cv2.CascadeClassifier('./cascade_files/ \
                                        haarcascade_frontalface.xml')
while True:
    #Llegim la imatge del dron
    frame_read = drone.get_frame_read()
    myFrame = frame_read.frame
    img = cv2.resize(myFrame, (width, height))
    #Obtenim el contorn del rostre detectat
    getContours(img)
```

Codi 5.6: Programa principal detecció cares amb *Cascade Classifier*

6 Llaç de control

L'últim pas previ per assolir l'objectiu d'aquest treball és definir un llaç de control que actuï sobre el dron amb la informació obtinguda en l'apartat 5 i aconseguir un seguiment òptim i continu.
A continuació es presenten les dues opcions triades per dur a terme aquesta tasca:

1. Control basat en regions

2. Control basat en IBVS

Recordar que la informació d'entrada dels mètodes de control és un punt en el pla de la imatge a part de les dades que ja coneixem de l'objecte. Dades que ens poden aportar informació de la distància a la que es troba l'objecte. De la mateixa manera que en l'anterior apartat, a mesura que es presenten els diferents mètodes de control també es mostra el resultat de la implementació.

També s'han de tenir en compte les situacions en les quals el sistema no troba detecció. En aquests casos s'ha optat per realitzar un moviment de rotació sobre si mateix "*yaw*" durant un temps determinat obtingut a partir de les frames sense detecció d'objectiu. Quan s'assoleixen aquests nombre de frames, s'executa un aterratge controlat per tal de finalitzar el control. En funció de l'utilitat final del sistema, es pot implementar una o altra resposta enfront de la falta de detecció.

A continuació s'expliquen les diferents opcions presentades per quan hi ha detecció.

6.1 Mètode de control basat en regions

La idea d'aquest primer mètode és molt senzilla. Consisteix en dividir el pla de la imatge en diferent zones per tal d'actuar de certa manera en funció d'en quina zona es troba l'objecte a seguir. En aquest treball l'objectiu sempre serà centrar l'objecte al centre del pla de la imatge.
Les regions que s'han considerat més òptimes per al correcte desenvolupament del dron després de diverses proves són les següents:

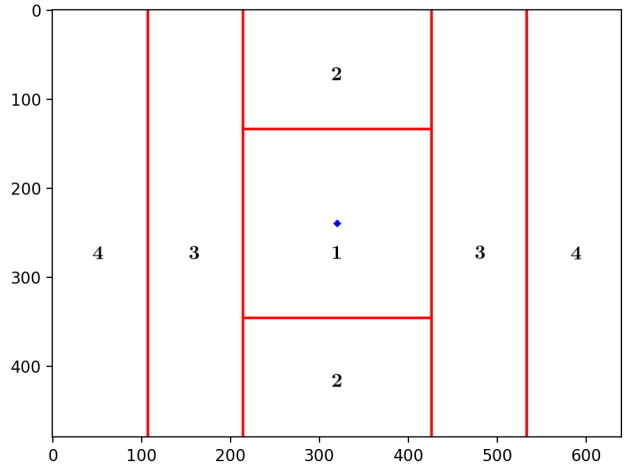


Figura 6.1: Zones d'actuació respecte el pla d'imatge

El punt blau representa el centre del pla de la imatge. La zona que l'engloba, la zona 1 a

l'anterior figura, s'ha considerat com una zona de marge d'error on considerem que el dron està en la posició correcte així que el senyal de correció és nul·la.

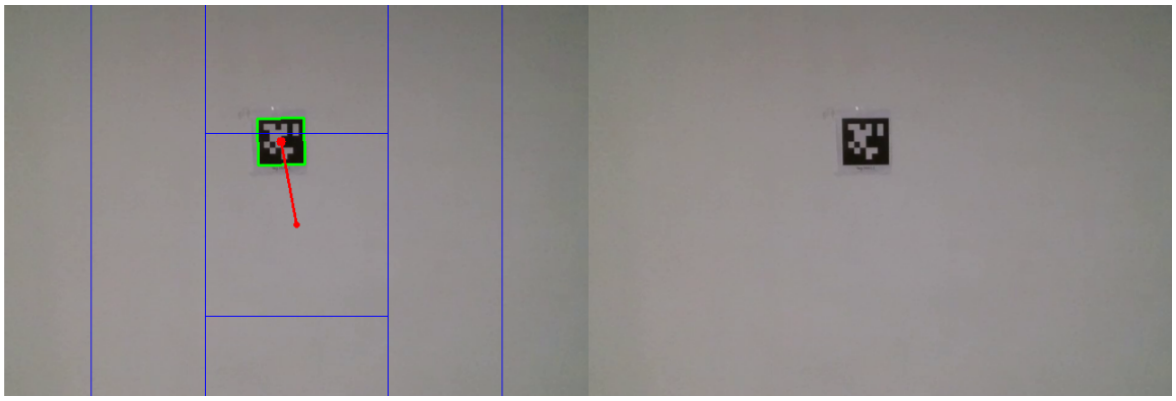


Figura 6.2: Control per regions, zona 1

En la figura 4.1 podeu recordar el sistema de coordenades definit pel sistema del dron. On el moviment horitzontal està governat per la coordenada x, el vertical per la y i per últim, el moviment endavant i enrere està controlat per la coordenada z. Les zones 2 situades a la part superior i inferior de la zona considerada com correcte són les zones encarregades de controlar la direcció y del dron. En cas que ens trobem en la zona 2 superior, el dron rebrà el senyal d'augmentar la seva velocitat vertical. En cas que estigui en la zona 2 inferior realitzarà la tasca oposada.

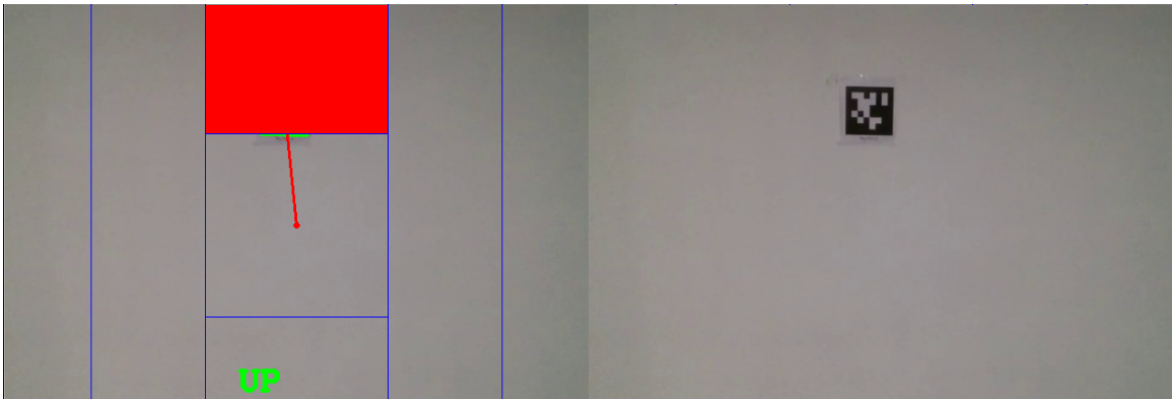


Figura 6.3: Control per regions, zona 2

Respecte a les zones 3 s'ha optat per realitzar un moviment de rotació respecte l'eix vertical, és a dir, un moviment de *yaw*. En cas que l'objectiu estigui localitzat a la zona 3 esquerra, el dron realitzarà un moviment antihorari. En cas contrari, moviment horari.

Figura 6.4: Control per regions, zona 3

Per últim, les zones 4 situades als dos extrems s'encarreguen dels moviments laterals. En cas que el *target* es trobi en una d'aquestes zones el programa envia una senyal al dron per augmentar la velocitat cap a la seva esquerra o dreta en funció de si es troba a la zona de l'esquerra o de la dreta.

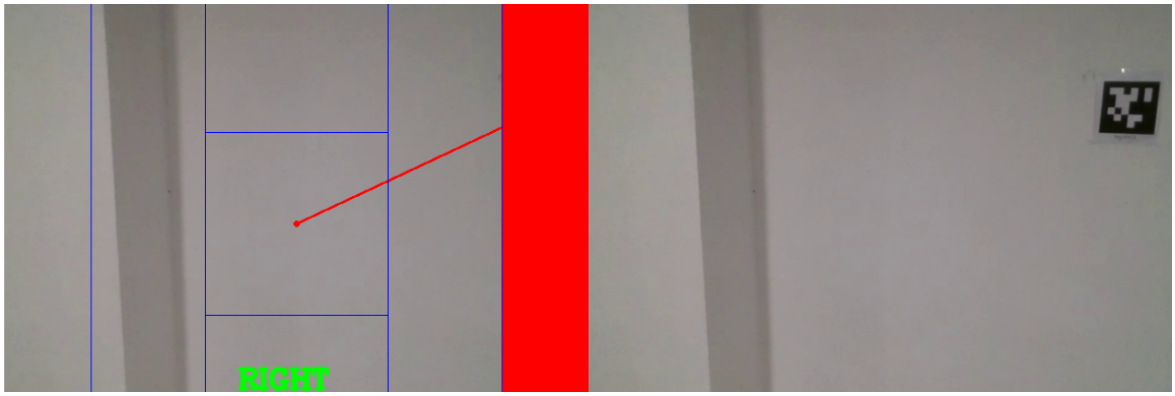


Figura 6.5: Control per regions, zona 4

A més a més, aquest control permet un moviment en la direcció z del dron. És a dir, endavant i enrere. Aquest control es realitza de diferents maneres en funció del *target* a seguir. En cas dels *AprilTags*, al ser de dimensió coneguda podem calcular la distància a la qual es troba amb gran facilitat. Per als altres mètodes de detecció, aquest control es realitza amb un petit càlcul previ. L'usuari ha d'especificar en quin rang d'àrees vol que ocupi el seu objecte en el pla de la imatge.

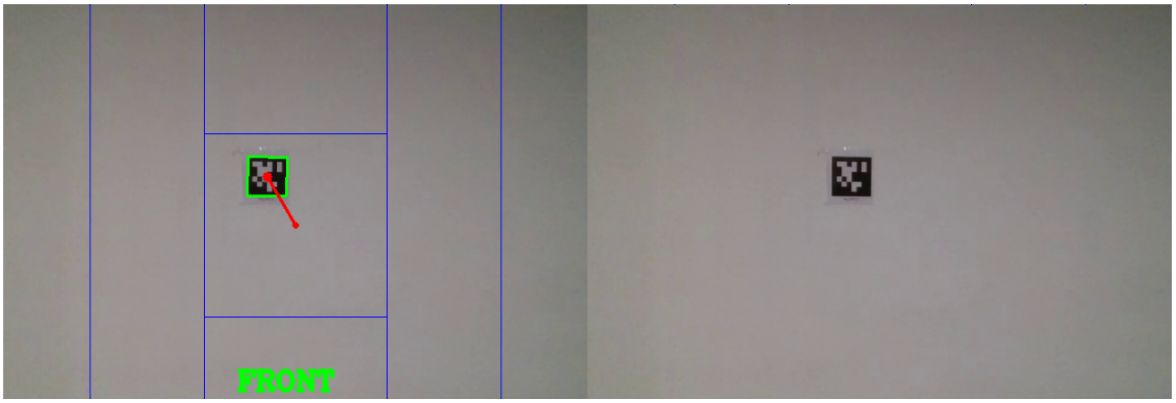


Figura 6.6: Control per regions, moviment endavant

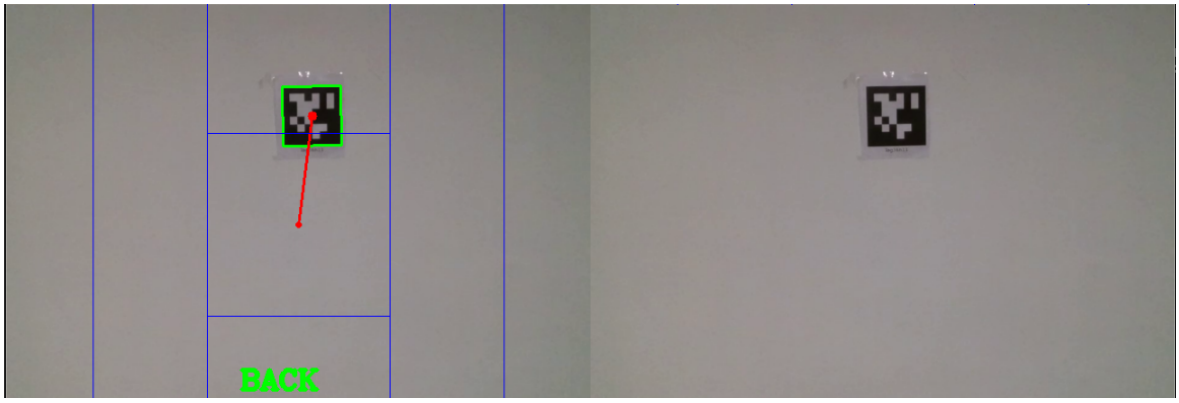


Figura 6.7: Control per regions, moviment enrere

Durant les proves dels moviments endavant i enrere s'ha comprovat que al no tenir en compte en quina zona del pla de la imatge estava l'objecte abans de realitzar aquests moviments, pot induir a una pèrdua de detecció degut al camp de visió. Per això, s'ha optat per combinar els moviments endavant i enrere amb un vertical en funció de en quina zona es troba l'objecte a l'hora de realitzar aquests moviments. La idea és, per exemple, si el target es troba a una distància superior a la màxima i per la zona inferior del pla de la imatge el dron realitzarà un moviment cap endavant i un moviment vertical proporcional a la velocitat d'avenç tenint en compte l'angle format respecte al punt principal del pla de la imatge (centre). Aquesta implementació la podeu veure entre les línies 1 i 20 del codi 6.2.

Per aquest control en concret, l'usuari pot indicar mitjançant un *slider* a quina velocitat vol que es realitzi el seguiment:

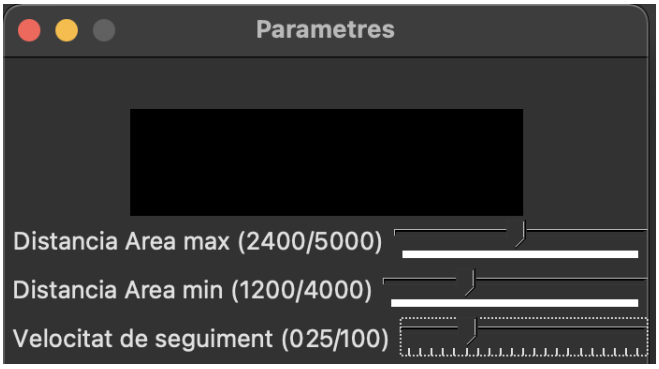


Figura 6.8: Control de paràmetres per l'usuari

Com es pot veure a la taula 3.2, aquest dron disposa d'un rang de velocitats entre -100 i 100 cm/s així que l'usuari pot triar entre un rang entre 0 i 100 cm/s per a realitzar el control. L'algorisme seqüencial seguit per aquest control a partir del centre de l'objecte és el següent:

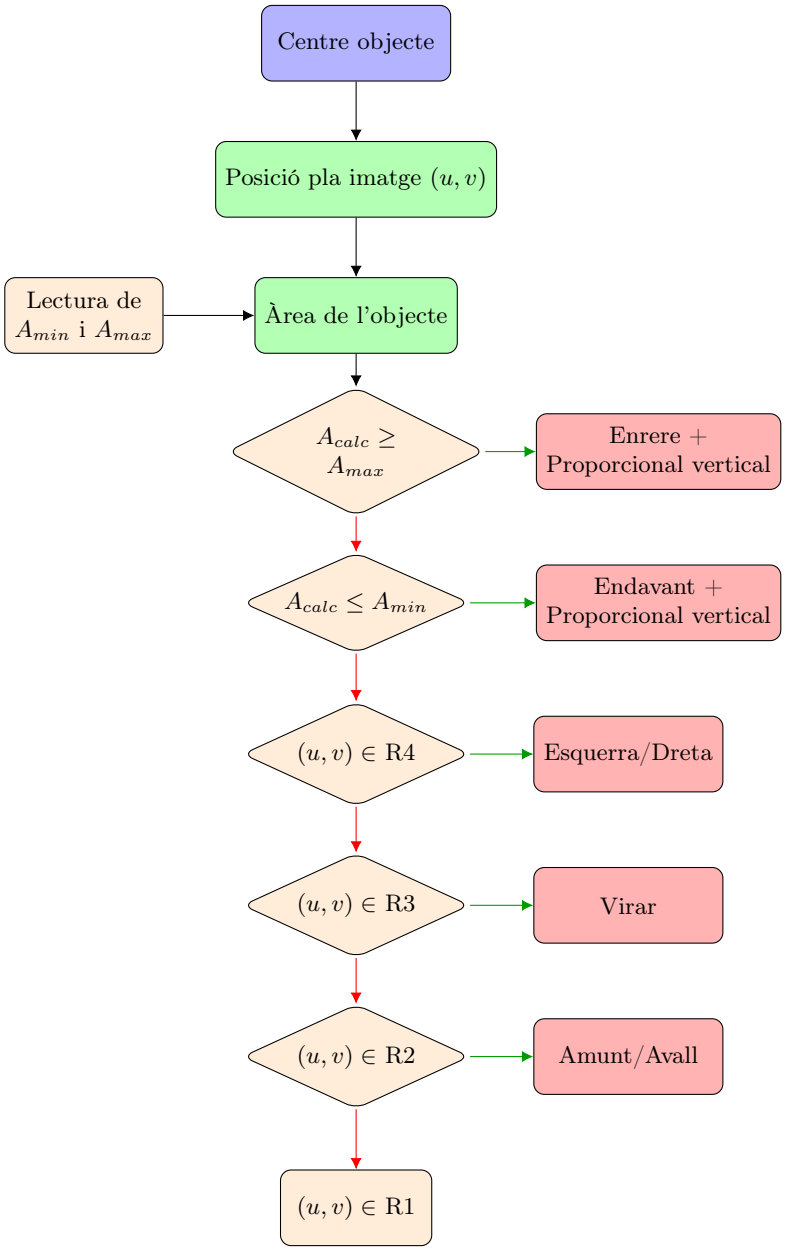


Figura 6.9: Algorisme actuació per regions

A continuació es mostra una figura amb la regla seguida pels algorismes del present treball:

A continuació podeu veure el codi implementat en *Python*:

```python
#Decisio de direccio en funcio de la informacio obtinguda dels programes de
    deteccio
# -- AREA -- Cx -- Cy
#Procedim a realitzar el algorisme explicat i a dibuixar els resultats en l'
    imatge
   if deteccio:
        area = cv2.contourArea(cnt)
        areaMin = cv2.getTrackbarPos("Area", "Parameters")
        if area > areaMin:
            if (area < cv2.getTrackbarPos('Distancia Area min', 'Parameters')):

                dir = 1 # Aqui definim la direccio
            elif (area > cv2.getTrackbarPos('Distancia Area max',
                                            'Parameters')):
                dir = 2
            elif (cx < int(frameWidth/2)-deadZone2):

                dir = 3
            elif (cx > int(frameWidth/2)+deadZone2):

                dir = 4
            elif (cx < int(frameWidth/2)-deadZone):

                dir = 5
            elif (cx > int(frameWidth / 2) + deadZone):

                dir = 6
            elif (cy < int(frameHeight / 2) - deadZone):

                dir = 7
            elif (cy > int(frameHeight / 2) + deadZone):

                dir = 8
            else:
                dir = 0 # Ens trobem dins dels marges de deteccio

     else:
          dir = 9 #Area inferior a la marcada, ha de seguir buscant
    else:
      dir = 9 #No hi ha detecio
```

Codi 6.1: Control basat en regions. Detecció regió

```python
#En base a la 'dir' acutem segons l'algorisme actuat i la velocitat
v = cv2.getTrackbarPos('Velocitat', 'Parameters')
   if dir == 1:
        drone.for_back_velocity = v
        if cy - frameHeight//2 > 0:
            drone.up_down_velocity = (- v*abs(cy-frameHeight//2)
                                      // (frameHeight//2))
        else:
            drone.up_down_velocity = (+ v*abs(cy-frameHeight//2)
                                      // (frameHeight//2))
        drone.left_right_velocity = 0
        drone.yaw_velocity = 0
    elif dir == 2:
        drone.for_back_velocity = -v
        if cy - frameHeight//2 > 0:
            drone.up_down_velocity = (v*abs(cy-frameHeight//2)
```

```
                                        // (frameHeight//2))
        else:
            drone.up_down_velocity = (- v*abs(cy-frameHeight//2)
                                        // (frameHeight//2))
        drone.left_right_velocity = 0
        drone.yaw_velocity = 0
    elif dir == 3:
        drone.left_right_velocity = -v
    elif dir == 4:
        drone.left_right_velocity = v
    elif dir == 5:
        drone.yaw_velocity = -v
    elif dir == 6:
        drone.yaw_velocity = v
    elif dir == 7:
        drone.up_down_velocity = v
    elif dir == 8:
        drone.up_down_velocity = -v
    elif dir == 9:
        drone.yaw_velocity = v
    else:
        drone.left_right_velocity = 0
        drone.for_back_velocity = 0
        drone.up_down_velocity = 0
        drone.yaw_velocity = 0
    #Enviem les velocitats al dron
    if drone.send_rc_control:
        drone.send_rc_control(drone.left_right_velocity, drone.for_back_velocity
    ,
                              drone.up_down_velocity, drone.yaw_velocity)

```

Codi 6.2: Control basat en regions. Actuació.

6.2 Mètode de control basat en algorisme *IBVS*

El segon i últim mètode de control dissenyat és un control basat en la teoria explicada en l'apartat 4.4.1.3. Com s'ha comentat, aquesta teoria utilitza la pseudoinversa del anomenat *Jacobià* de la imatge per, mitjançant un controlador proporcional, poder trobar quina és la velocitat de la càmera sabent la posició objectiu de l'objecte a seguir.

La fórmula que governa aquest algorisme, explicada a l'apartat 4.4.1.3, és la següent:

$$\boxed{\dot{\boldsymbol{r}} = \phi \left[\boldsymbol{J}(\boldsymbol{p1}, Z)\right]^{+} (f^{*} - f)} \tag{24}$$

Per poder resoldre aquesta equació s'han de tenir en compte diferents consideracions. Com es pot veure a l'equació 14 el *Jacobià* es calcula a partir de la distància focal (en unitats de píxels), la posició en el pla de la imatge i la distància del pla de la imatge a l'objecte. A més a més, com es pot veure en l'anterior equació, també existeix el controlador lineal, ϕ, el qual hem d'entrenar per al nostre sistema.
A continuació expliquem un a un com s'han solucionat els aspectes comentats:

1. **Distància focal en píxels**
   El fabricant ens proporciona la distància focal en mil·límetres. Per tal de trobar la distància focal en píxels només era necessari dividir entre la densitat mitjana de píxels de la càmera. Com no disposem d'aquesta informació hem trobat aquest valor amb càlculs simples, com el *Teorema de Tales*. Mitjançant similituds de triangles i la següent fórmula podem trobar la distància focal en píxels:
   $$\lambda' = \frac{PZ}{W} \tag{25}$$
   on λ' és la distància focal en píxels, P és el nombre de píxels que ocupa l'objecte en el pla de la imatge, W la dimensió de l'objecte en l'eix a estudiar i per últim, Z, la distància de la càmera a pla de la imatge. Amb aquest simple experiment realitzat amb un objecte de dimensions conegudes, com per exemple un marcador visual *AprilTag*, s'ha trobat la distància focal del dron *DJI Tello*
   $$\lambda' = 592.59\ pxels \tag{26}$$

2. **Distància a l'objecte**
   De la mateixa manera que en l'anterior punt, un cop tenim la distància focal de la càmera i també coneixem les dimensions de l'objecte, també a través del *Teorema de Tales* podem trobar molt fàcilment a quina distància es troba l'objecte:
   $$Z = \frac{W\lambda'}{P} \tag{27}$$
   Aquesta equació s'utilitza en l'algorisme seguit un cop per cicle per a trobar la distància a la que es troba l'objecte per poder calcular el *Jacobià*.

3. **Controlador lineal ϕ**
   Per últim, s'ha realitzat el control lineal. L'objectiu d'aquest entrenament és trobar un valor de ϕ capaç de trobar una velocitat òptima per a realitzar el control a qualsevol punt del pla de la imatge.
   Entre les diferents metodologies que existeixen la decidida per trobar aquest valor és mitjançant una tècnica anomenada *Teach by showing* que consisteix en obtenir el valor desitjat de ϕ a diferents punts i després calcular una mitjana ponderada. El procediment seguit consisteix en els següents passos:

   - Primer es resol la següent equació a diferents posicions del pla de la imatge a distàncies pròximes a les quals es vol fer el control real.
     $$\phi = \frac{\dot{\boldsymbol{r}}}{\Big[\boldsymbol{J}(\boldsymbol{p1}, Z)\Big]^{+}(f^* - f)} \tag{28}$$
     En funció de les velocitats, $\dot{\boldsymbol{r}}$, triades el resultat serà un o un altre.
   - Un cop tenim calculats els diferents valors del controlador ϕ per al ventall d'opcions més òptim, procedim a calcular un valor mitjà ponderat:
     $$\phi = \sum_{i=1}^{N} \frac{\phi_i}{N} \tag{29}$$

El valor obtingut que millor s'adequa al control en les nostres condicions és:

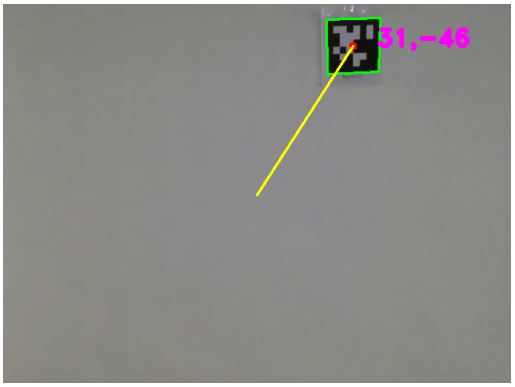
$$\phi = 500.000 \tag{30}$$

Una altra manera d'entrenar aquest controlador podria ser mitjançant una discretització del pla de la imatge per tal d'utilitzar un controlador diferent en funció de la zona on es trobi que optimitzi el seu funcionament.

En el present treball només s'ha realitzat l'entrenament del controlador lineal amb la tecnologia *AprilTag*. De totes maneres, aquest entrenament es pot realitzar amb qualsevol objecte de dimensions conegudes.

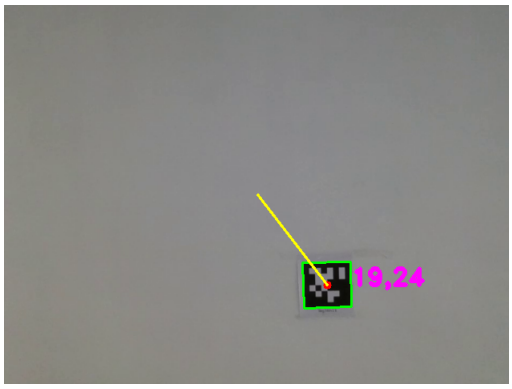
En l'actualitat existeixen diverses línies d'investigació pel control d'un dron sense calibratge com podeu veure a [10].

A continuació es presenten un seguit de figures amb els resultats de la implementació d'aquest sistema de control:



(a) Zona dreta superior

(b) Zona esquerra superior



(c) Zona dreta inferior

(d) Zona esquerra inferior

Figura 6.10: Exemples d'actuació control en diferents punts

Com es pot comprovar en les anteriors figures, en funció del quadrant en el qual es troba el objectiu, el signe de la velocitat canvia, ja que el centre de referència, és a dir el punt de canvi, es troba al centre del pla de la imatge. També es pot comprovar que quant més allunyat del centre es troba més augmenta la velocitat, ja que intenta arribar a l'objectiu en el mateix espai de temps.

En la següent figura veiem com quan l'objectiu es troba a prop del centre, les velocitats són molt petites:

Figura 6.11: Example control a prop del objectiu

Un cop l'objectiu és assolit, les velocitats obtingudes per el model són 0:

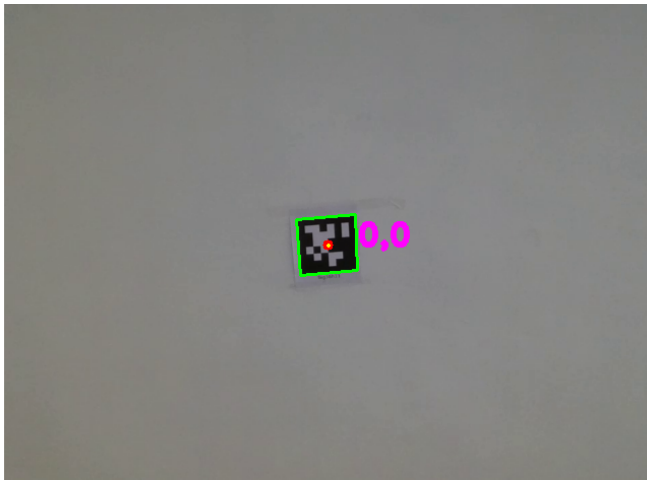


Figura 6.12: Exemple control objectiu assolit

Per aconseguir un correcte funcionament de l'algorisme i de la mateixa manera que en el llaç de control anterior, s'ha definit una zona quadrada (R) al voltant del centre del pla de la imatge per estabilitzar el control al voltant d'uns llindars. Al ser un dispositiu volador subjecte a tant soroll de l'exterior, és molt difícil mantenir la posició amb només un píxel de control.

A continuació es presenta l'algorisme seguit:

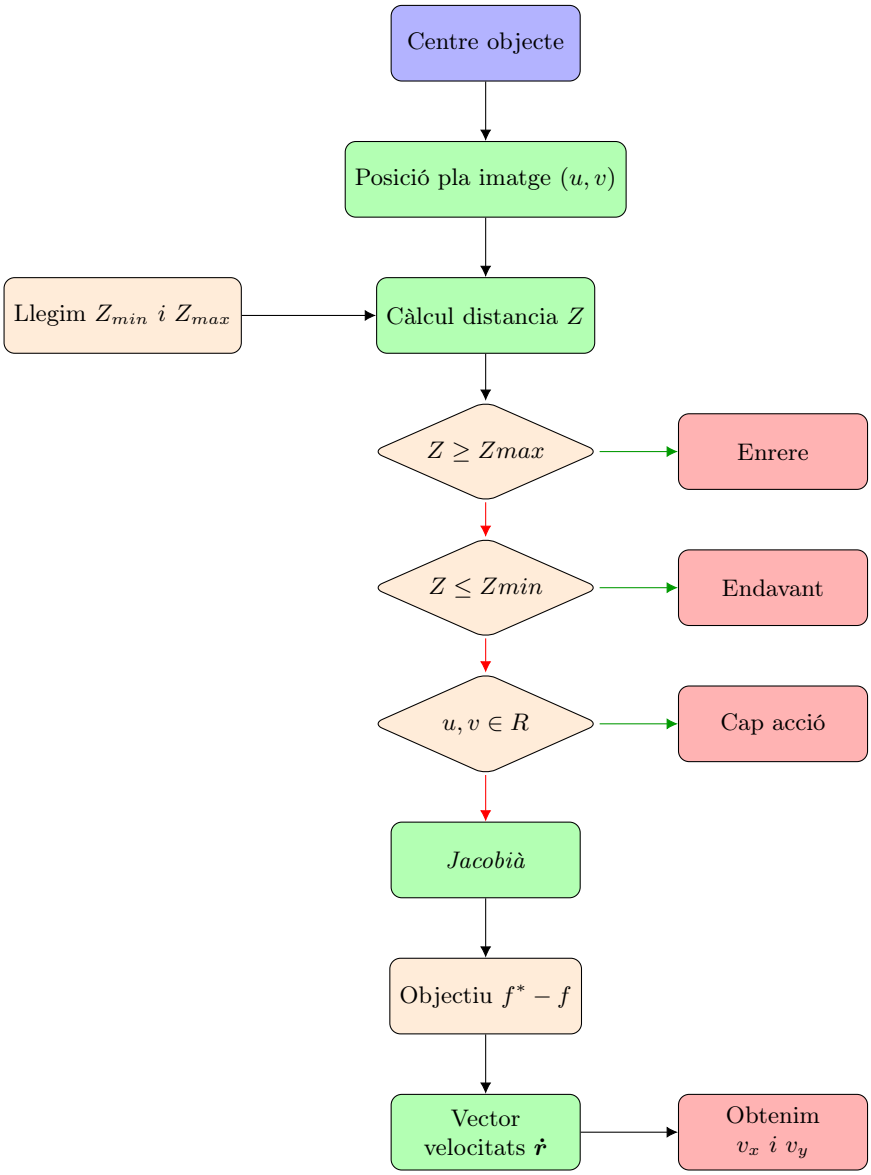


Figura 6.13: Algorisme control basat en *IBVS*

A continuació podeu veure el codi implementat en *Python*:

```
#Definicio de la funcio
def getJacobian(cx, cy, Z):
    global f
    global lamda
    #Obtenim les coordenades del target respecte el 'principal point'
    u = cx - frameWidth//2
    v = cy - frameHeight//2
    #Calcul del Jacobia
    J = np.array([[-f/Z, 0, u/Z, u*v/Z, -(f*f+u*u/f), v],
                  [0, -f/Z, v/Z, -(f*f+v*v/f), -u*u/f, -u]])
    #Obtenim la ivnersa
    pJ = np.linalg.pinv(J, rcond=1e-15, hermitian=False)
    #Calculem l'objectiu a assolir
    p_star = np.array([[0-u], [0 - v]])  # Objectiu: El centre de la imatge
```

```python

    #Calculem les velocitats
    v = 100 * lamda * np.matmul(pJ, p_star)

    #Limitem les velocitats del dron
    for i in len(v):
        if v[i] > 100:
            v[i] = 100
        elif v[i] < 100:
            v[i] = -100
    return v
```

Codi 6.3: Funció per obtenir el Jacobià

```python
v_user = cv2.getTrackbarPos('Velocitat', 'Parameters')
#Programa principal
    #Comprovem que hi ha deteccio
    if detect == 1:
        if Z >= cv2.getTrackbarPos('Distancia max', 'Parameters')):
            drone.for_back_velocity = v_user
             if cy - frameHeight//2 > 0:
             drone.up_down_velocity = (- v_user*abs(cy-frameHeight//2)
                                       // (frameHeight//2))
              else:
                drone.up_down_velocity = (+ v_user*abs(cy-frameHeight//2)
                                      // (frameHeight//2))
            drone.left_right_velocity = 0
            drone.yaw_velocity = 0
        elif Z <= cv2.getTrackbarPos('Distancia min', 'Parameters'))::
            drone.for_back_velocity = -v_user
            if cy - frameHeight//2 > 0:
              drone.up_down_velocity = (v_user*abs(cy-frameHeight//2)
                                        // (frameHeight//2))
            else:
             drone.up_down_velocity = (- v_user*abs(cy-frameHeight//2)
                                      // (frameHeight//2))
             drone.left_right_velocity = 0
             drone.yaw_velocity = 0
        else:
              #Comprovem si esta dintre o fora del marge vertical
            if abs(cx-frameWidth//2) >= deadZone:
                drone.for_back_velocity = 0
                drone.left_right_velocity = int(v[0])
                drone.yaw_velocity = 0
                drone.up_down_velocity = 0
            else:
                drone.for_back_velocity = 0
                drone.left_right_velocity = 0
                drone.yaw_velocity = 0
                drone.up_down_velocity = 0
     #Comprovem si esta dintre o fora del marge vertical
            if abs(cy-frameHeight//2) >= deadZone:
                drone.for_back_velocity = 0
                drone.yaw_velocity = 0
                drone.up_down_velocity = -int(v[1])
            else:
                drone.for_back_velocity = 0
                drone.yaw_velocity = 0
                drone.up_down_velocity = 0
    else:
```

```
47          drone.for_back_velocity = 0
48          drone.left_right_velocity = 0
49          drone.yaw_velocity = 0
50          drone.up_down_velocity = 0
51      #Enviem les senyals al dron
52      if drone.send_rc_control:
53          drone.send_rc_control(drone.left_right_velocity,
54                                drone.for_back_velocity,
55                                drone.up_down_velocity, drone.yaw_velocity)
```

Codi 6.4: Programa principal actuació basat en *IBVS*

7 Discussió de Resultats

Abans d'agrupar els diferents sistemes per obtenir un sistema de control complet procedim a realitzar un petit estudi d'avantatges i inconvenients dels diferents mètodes presentats.

7.1 Detecció de color

Avantatges	Inconvenients
Baix cost computacional	Problemes de detecció a causa de factors ambientals
Fàcil implementació	Confusió amb altres colors a l'espai de treball
Bona continuitat sota condicions controlades	

Taula 7.1: Pros i contres de la detecció per color

7.2 Detecció *AprilTag*

Avantatges	Inconvenients
Molt bona capacitat de detecció	Necessitat de portar el *AprilTag* a sobre
Fàcil implementació	Possible confusió amb un altre *AprilTag* igual (Baixa probabilitat)
Baix cost computacional	Visibilitat del *AprilTag*
Grans distàncies	

Taula 7.2: Pros i contres de la detecció per *AprilTag*

7.3 Detecció *Cascade Classifier*

Avantatges	Inconvenients
Fàcil implementació	Problemes de detecció degut a moviments
Baix cost computacional	Pot portar problemes de confusió en certes situacions
No necessita cap objecte	

Taula 7.3: Pros i contres de la detecció per *Cascade Classifier*

7.4 Control basat en regions

Avantatges	Inconvenients
Intuïtiu	Vàlid tant sols per a cert rang de distancies
Bona capacitat de control	

Taula 7.4: Pros i contres de la detecció per control per regions

7.5 Control basat en IBVS

Avantatges	Inconvenients
Seguiment ràpid	Difícil calibració del llaç de control
Bona capacitat de control	Calibració vàlida per certes distàncies
	Necessitat d'informació de l'objecte a seguir

Taula 7.5: Pros i contres de la detecció per control IBVS

Un canvi de distància de treball notori hauria de comportar un canvi de regions d'actuació o una recalibració del llaç de control

7.6 Combinacions

Finalment s'han agrupat les diferents metodologies presentades, tant de detecció com de control en diferents combinacions per tal de realitzar una petita comparativa de la *performance*. Les combinacions realitzades són les següents:

- Detecció color amb control basat en regions
- Detecció *AprilTag* amb control basat en regions
- Detecció *Cascade Classifier* amb control basat en regions
- Detecció *AprilTag* amb control *IBVS*

Aquestes combinacions s'han obtingut tenint en compte la informació que hem de tenir de cada objecte per alimentar al sistema de control.

A continuació es presenta un estudi de les diferents combinacions presentades per tal de comprovar quin és el comportament de cada una enfront un canvi en la posició de l'objectiu.
Per tal d'obtenir aquestes figures s'han realitzat una sèrie de moviments arbitraris en diferents sentits dintre del camp de visió del dron i fora de la zona segura. L'error es defineix com la distància entre l'objectiu a seguir i el centre del pla de la imatge en píxels. Matemàticament

l'error es pot definir de la següent forma:

$$\boldsymbol{e} = \sqrt[2]{\left(\frac{W}{2} - c_x\right)^2 + \left(\frac{H}{2} - c_y\right)^2} \tag{31}$$

on W és l'ample de la imatge, H l'altura i (c_x, c_y) són les coordenades del centre de l'objectiu. Aquest error definit està acotat segons:

$$\boldsymbol{e} = \left[0, \sqrt[2]{\frac{H^2}{4} + \frac{W^2}{4}}\right] = [0, 400] \tag{32}$$

Per tal de suavitzar el seguiment del dron i com s'ha comentat a la secció 6, s'ha decidit deixar un marge d'actuació del 25% en el qual si l'objectiu es troba dins d'aquest marge l'algorisme ho entendrà com una posició vàlida, és a dir, intentarà mantenir-la. Aquest marge es pot veure representat en les següents figures com una línia discontinua de color verd.

Detecció color amb control basat en regions

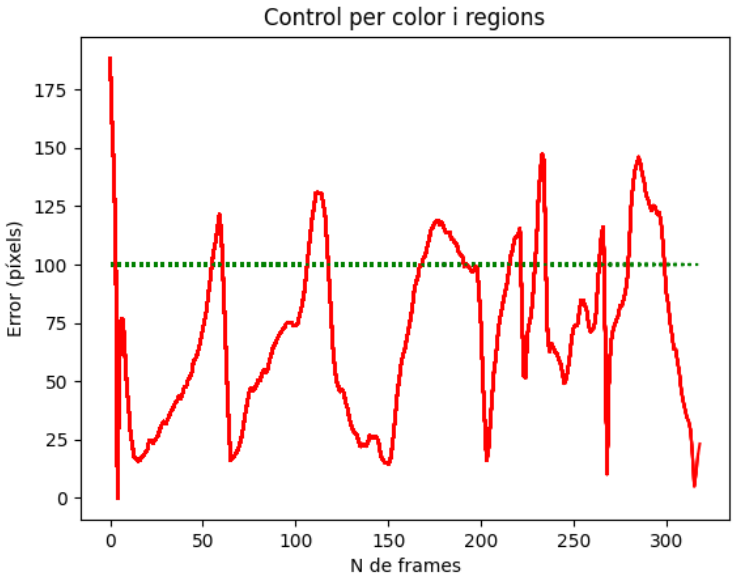


Figura 7.1: Performance detecció color i control basat en regions

En l'anterior figura podem veure l'evolució de la distància (en píxels) entre el centre de l'objecte i el centre del pla de la imatge. La línia discontïnua verda indica la frontera a la qual es considera correcte el control si la distància entre centres és menor que aquesta. En cas contrari, es recalibra la posició. Es pot veure que per a cada pic, el retorn a la zona segura és molt ràpid, pràcticament igual que el moviment desestabilitzador. També podem veure que, sota condicions controlades, la detecció del color és continua i sense errades.
Els moviments generats per sota de la zona segura són moviments inevitables deguts a la dinàmica del dron i l'entorn de treball.

Detecció *AprilTag* amb control basat en regions

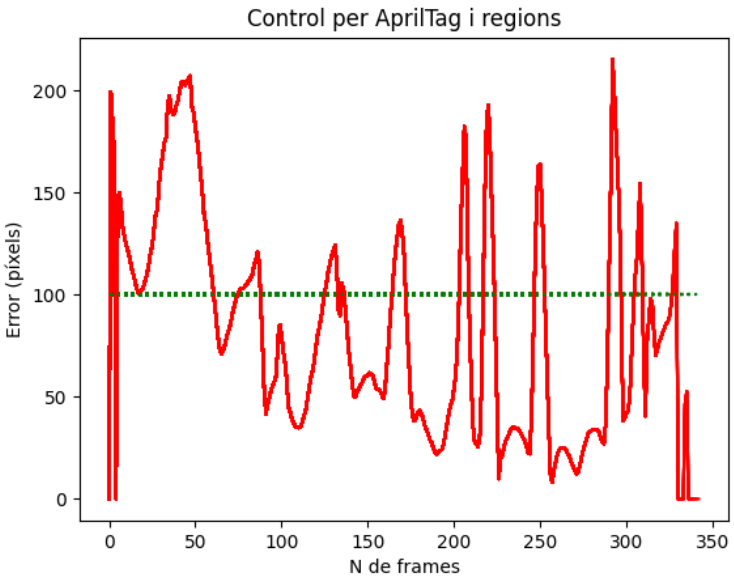


Figura 7.2: Performance detecció *AprilTag* i control basat en regions

Respecte a la detecció amb *AprilTag* i el control basat en regions podem veure que la resposta davant d'un canvi és molt més ràpida que la detecció per color. Veiem que els canvis tenen més pendent. Tampoc hi ha cap moment de pèrdua de detecció, un punt molt important a tenir en compte per assegurar un correcte funcionament.

Detecció *Cascade Classifier* amb control basat en regions

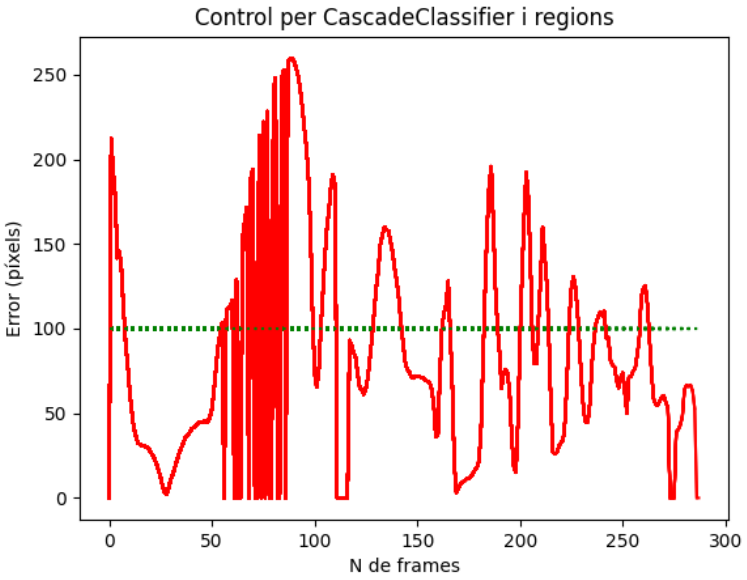


Figura 7.3: Performance detecció *Cascade Classifier* i control basat en regions

Com es pot veure en l'anterior figura, la detecció amb l'algorisme *Cascade Classifier* té un bon retorn a la zona vàlida cada cop que es produeix una pertorbació. També podem veure que hi ha diverses pèrdues de detecció entorn les frames 50 i 100. Això és degut a que el model de detecció utilitzat únicament detecta el frontal del rostre d'una persona i els moviments d'aquesta amb el cap poden afectar a la detecció. Tot i així el dron és capaç de mantenir el control i oferir una bona continuïtat.

Detecció *AprilTag* amb control basat en IBVS

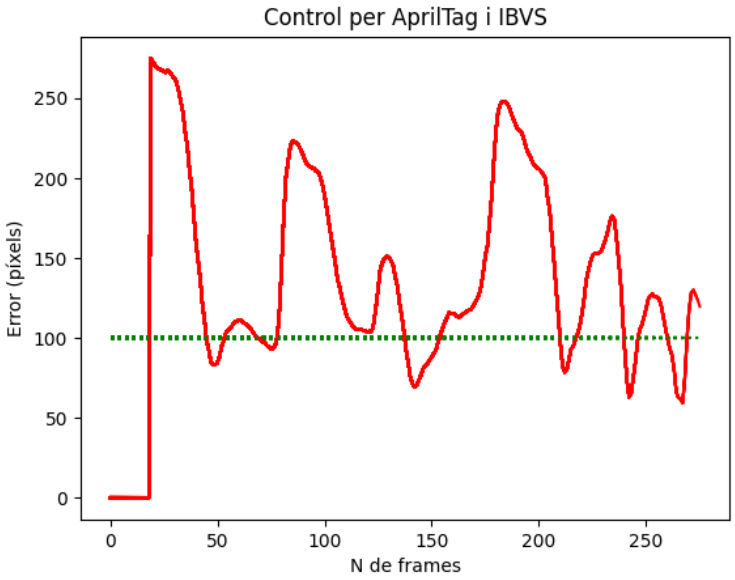


Figura 7.4: Performance detecció *AprilTag* i control basat IBVS

El resultat de la implementació del *Jacobià* és satisfactòria, ja que assoleix l'objectiu proposat de seguiment. Tot i això, és un sistema que presenta alguns problemes que es poden veure reflectits en l'anterior figura. L'algorisme, com podeu veure, utilitza la dimensió en píxels del *AprilTag* per extreure a quina distància es troba. Com és d'esperar, les rotacions de la marca visual afecten el sistema, ja que l'entrada és una distància errònia. Si les rotacions són continues el control acaba sent molest per la gran quantitat de moviments que realitza el dron tot i que efectiu en un espai ampli.

De totes maneras i tal com es diu en els objectius s'ha intentat que cap algorisme complet tingués un cost computacional molt alt ja que derivaria en un temps de cicle també alt i no respectaríem el marge entre 2 i 20 Hz definit als objectius. Per comprovar-ho, s'ha realitzat un estudi per obtenir el màxim temps d'execució per cada una de les combinacions presentades. Els resultats són els següents:

- Detecció color amb control basat en regions

$$t \in (0.061, 0.1523)\ s \tag{33}$$

- Detecció *AprilTag* amb control basat en regions

$$t \in (0.065, 0.1377)\ s \tag{34}$$

- Detecció *CascadeClassifier* amb control basat en regions

$$t \in (0.083, 0.1419)\ s \tag{35}$$

- Detecció *AprilTag* amb control basat en IBVS

$$t \in (0.064, 0.1339)\ s \tag{36}$$

Gràcies a les diferents proves realitzades, es pot assegurar que cap programa de control complet té un període de execució superior a 160 mil·lisegons, permetent una resposta de sistema continua i efectiva envers a canvis de posició de l'objectiu dintre del pla de la imatge. Cal recordar que el fabricant recomana un temps entre intercanvi de paquets amb el dron de, mínim, 500 mil·lisegons.

$$f_{exec} \in (6.5,\ 16)\ Hz \tag{37}$$

Un estudi més exhaustiu respecte a la rapidesa de reacció del dron es podria haver generat realitzant una neteja de les dades en la que només ens quedem amb els punts que constitueixen una recta de pendent negatiu i amb valor superior al valor límit d'acció, en aquest cas 100, amb l'objectiu d'extreure un pendent mitjà i poder comparar numèricament els comportaments. De totes maneres, amb l'estudi visual realitzat ja és suficient per poder comparar les combinacions, ja que l'experiment no està compost pels mateixos moviments per a tots els estudis.

Per últim comentar que el sistema és estable enfront de petites pertorbacions com podria ser la pèrdua de detecció momentani gràcies a un comptador. Aquest comptador permet perdre la detecció fins a un màxim de 10 imatges (*frames*). En cas de superar aquest llindar, el dron realitzarà un moviment de *yaw* durant un temps fixat, ja que entendrà que ha perdut la detecció i intentarà buscar un altre objectiu. Tanmateix, si supera aquest altre llindar, es realitzarà un aterratge immediat. Tant aquesta implementació com les altres es poden veure a l'annex adjunt. Per afegiment, el dron seleccionat disposa d'un sistema de control que mitjançant el sensor de posicionament, que podeu veure a la taula 3.1, és capaç d'augmentar o disminuir la velocitat si detecta algun obstacle proper. Aquest sistema té preferència enfront del nostre llaç de control.

8 Impacte ambiental

L'augment en l'aplicació de sistemes servo control en la indústria gènera un impacte positiu al medi ambient, ja que aquests sistemes són capaços de suplir gran varietat dels sensors que s'utilitzen avui en dia. De la mateixa manera, a part de poder assolir les tasques acomplides per altres sensors, també són capaços de disminuir el nombre de sensors utilitzats per dur a terme les mateixes funcions.

Per altra banda, l'augment en la utilització de vehicles aeris no tripulats també porta bones conseqüències al medi ambient. Aquesta afirmació, però porta matisos, ja que aquest augment ha d'anar acompanyat d'una regulació adequada. Es poden imaginar moltes aplicacions tals com el repartiment de correu a domicili, entrega de paquets de petites dimensions, suport mèdic a persones amb dificultats per sortir del domicili, etc. En l'actualitat totes les tasques comentades es realitzen amb vehicles de motor que podrien ser substituïts millorant així inclús l'eficàcia d'aquests processos.

Tal com s'ha comentat, la introducció del vehicle aeri no tripulat ha d'anar acompanyat d'una regulació que permeti assegurar el correcte funcionament així com la seguretat de l'usuari o de les persones que es trobi en el desenvolupament de la tasca.

9 Cost del projecte

S'ha realitzat un estudi econòmic del projecte tenint en compte les diferents despeses que se'n deriven. L'objectiu principal és realitzar una aproximació monetària del cost que podria significar un estudi com aquest.

El cost total d'aquest projecte es limita a la compra del Dron DJI Tello i a qualsevol dispositiu capaç de realitzar els càlculs computacionals duts a terme. La resta de tecnologies utilitzades són de lliure accés.

En aquest pressupost es té en compte les despeses de personal, corresponents a les hores de feina de l'enginyer que faria el projecte. Basant-se en les hores mínimes necessàries corresponents a 12 crèdits ECTS i observant posteriorment la quantitat de temps destinat en global, es considera un total aproximat de 375 hores per a completar l'estudi. Tenint en compte que l'estudi el realitza un enginyer, el preu és de 60 €/hora.

Per últim, es té en compte les despeses d'establiment d'aquest projecte. Se suposa que es realitza en un local. D'aquesta manera es pot considerar el lloguer del local, estipulat en uns 900 €/mes.

Amb l'objectiu de trobar el mínim cost possible, s'ha realitzat la següent taula amb els dispositius indispensables:

Material	-
Dron DJI Tello	120 €
Raspberry pi	35 €
Pantalla i Teclat	120 €
Personal Enginyer	- 22.500 €
Lloguer Alquiler	– 4.500 €
Cost total	**27.275** €

Taula 9.1: Cost del projecte

10 Planificació del projecte

A continuació es presenta la planificació temporal seguida per a la realització del projecte. El següent diagrama de Gantt s'ha discretitzat en dinou setmanes compreses entre el 7 de setembre del 2020 i el 18 de gener del 2021.

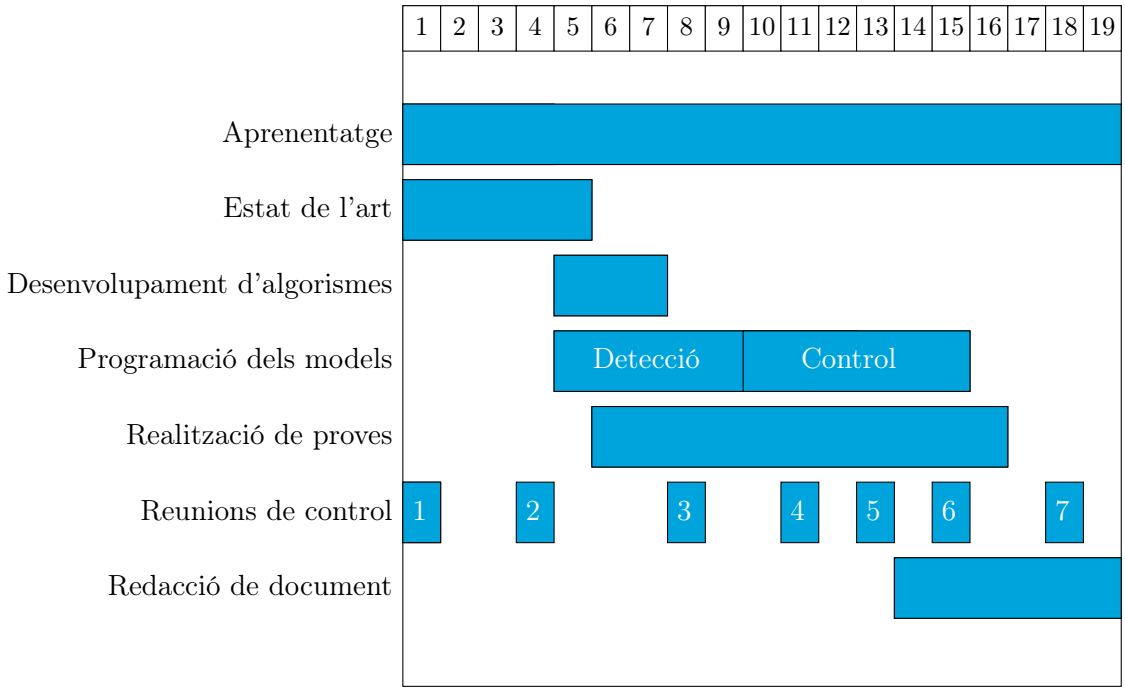


Figura 10.1: Diagrama de Gantt

Conclusions

Els resultats obtinguts al finalitzar el projecte s'ajusten als objectius plantejats a l'inici, és a dir, s'ha dissenyat un sistema de control autònom per a un dron comercial mitjançant tècniques servo control de baix cost computacional.

Respecte als mètodes escollits de detecció els resultats són els esperats. La detecció basada en color és suficientment bona quan es té un petit control de les variables ambientals com podria ser en una fàbrica o nau industrial. En cas de disposar de mala qualitat de vídeo també és interessant utiltzar l'espai de color HSL en el qual la saturació tendeix cap al gris a diferència del model HSV que tendeix cap al blanc, situació menys real. A més a més el color és una bona referència utilitzada àmpliament en aquest tipus d'aplicacions per poder separar l'objectiu del fons de la imatge. L'empresa DJI utilitza aquest tipus de tecnologia. La següent tecnologia escollida, les marques visuals de la categoria *AprilTag* assoleixen perfectament l'objectiu de detecció. Altrament, a l'utilitzar marques visuals de composició coneguda és relativament fàcil obtenir les matrius de translació i rotació, informació que després es pot utilitzar per realitzar el control. Per últim, la tecnologia del detector en cascada també assoleix l'objectiu desitjat, però porta un problema inherent, s'ha d'estar en una situació en la qual no hi hag més persones. En cas contrari, si hi ha una detecció més propera canviarà d'objectiu. Existeixen diferents maneres per solucionar aquest problema: la primera és implementar un reconeixement facial que permeti a l'usuari introduir una quantitat reduïda de fotos seves amb diferent il·luminació o implementar un identificador numèric a cada target detectat amb l'objectiu de realitzar el control sobre un identificador específic. Un altre aspecte a comentar respecte al classificador en cascada és que únicament detecta cares de manera frontal perdent la connexió cada cop que l'usuari realitza moviments rotatius amb el cap. Per solucionar aquest problema estaria bé implementar també un detector lateral de cares combinat amb un identificador numèric per no perdre la detecció en cap moment.

Aquest identificador utilitza un algorisme capaç d'emmagatzemar les posicions dels objectes detectats en l'anterior frame i comprovar mitjançant distància si és el mateix identificador o ha d'utilitzar un nou. Està subjecte a errors, però és una bona solució.

Pel que fa als algorismes de control ambdós han estat dissenyats per unes variables concretes d'entorn, és a dir, per a unes distàncies de seguiment fixes amb certa fulgura. Si es volgués canviar la distància de control s'haurien de replantejar les zones d'acció en funció de si s'augmenta o disminueix la distància i en el cas de l'algorisme IBVS tornar a realitzar un entrenament per a les diferents situacions. Una possible solució per a aquesta problemàtica podria ser implementar diferents models de zones o entrenament de control per a diferents distàncies així el programa serà capaç de discretitzar el tipus de control més òptim en funció de la distància a la qual es troba l'objectiu. Respecte al controlador utilitzat en l'algorisme IBVS, utilitza un controlador proporcional simple, una pròxima futura línia d'investigació podria ser la incorporació d'un control integral per tal de realitzar un control més agradable i suau.

Un altre futur treball a realitzar per tal de millorar aquesta implementació podria ser la combinació de les diferents tècniques explicades mitjançant un algorisme sensorial que permeti detectar en funció de diversos factors tals com la visibilitat, il·luminació o moviment quin és l'algorisme de detecció o control que millor s'ajusta a les condicions detectades.

Afegir que únicament les connexions amb el dron i la comanda per enviar el senyal que conté

la informació de la velocitat són específiques per el dron seleccionat. La resta de codi es pot utilitzar en qualsevol altre dron que permeti connexió o aplicació que es cregui necessari.

Personalment trobo molt important la investigació i el desenvolupament d'aplicacions i/o algorismes en el camp de la visió per computador i més en l'aplicació de vehicles autònoms no tripulats com són els drons, ja que són tecnologies en augment que poden aportar un gran canvi a la societat. Actualment ja existeixen diverses aplicacions com la distribució de paquets a domicili per l'empresa Amazon a algunes ciutats importants dels Estats Units o inclús drons equipats amb material mèdic per atendre urgències. Aquests sistemes poden arribar al punt desitjat mitjançant tecnologia GPS però, al punt final en el moment on es troben amb el client, necessiten capacitat sensorial per tal de poder oferir un millor servei sense posar en risc a l'usuari. No tinc cap dubte que la manera més econòmica i eficient és mitjançant els algorismes de visió per computador capaços d'adaptar-se a l'entorn.

Agraïments

A l'Antoni Grau, per les directives i la confiança.
Als meus pares, per tot.

Bibliografia

[1] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, 1996.

[2] P. Corke, *ROBOTICS, Vision and Control*. Springer, 2017.

[3] F. Chaumette and S. Hutchinson, "Visual servo control. I. Basic approaches," *IEEE Robotics and Automation Magazine*, 2006.

[4] J. Wang and E. Olson, "AprilTag 2: Efficient and robust fiducial detection," *IEEE International Conference on Intelligent Robots and Systems*, 2016.

[5] E. Olson, "AprilTag: A robust and flexible visual fiducial system," *Proceedings - IEEE International Conference on Robotics and Automation*, 2011.

[6] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, 2004.

[7] S. M. Abbas, S. Aslam, K. Berns, and A. Muhammad, "Analysis and improvements in apriltag based state estimation," *Sensors (Switzerland)*, 2019.

[8] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001.

[9] Mahdi Rezaei, "Creating a Cascade of Haar-Like Classifiers: Step by Step," *Univeristy of Auckland, Computer Science Department*, 2013.

[10] J. Andrade-cetto and A. Santamaria-Navarro, "Uncalibrated image-based visual servoing," *Institut de Robòtica IRI, CSIC-UPC, Spain*, 2019.

[11] F. Steven A, *Control Theory Tutorial*. California, Irvine USA: Springer, 2018.

[12] L. He, N. Yanqiu, L. Zhengzheng, and F. Xiang, "Quadcopter autonomous problem flight based on recognition image," *World Congress on Intelligent Control and Automation (WCICA)*, 2016.