Dídac Coll Pujals

Interfície basada en Kinect i Leap Motion per a la interacció home-màquina

TREBALL DE FÍ DE GRAU

Dirigit Albert Oller Pujol

Grau en Enginyeria Electrònica Industrial i Automàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2015

Índex

Índex de figures

Índex de taules

Acrònims

1 Introducció

Avui en dia, estem envoltats de tecnologia. A mesura que passa el temps, tot allò que ens en volta s'està automatitzant i apareixen noves tecnologies excepcionals i assequibles econòmicament capaces de millorar la nostra vida diària i realitzar projectes que fins fa poc creiem que era gairebé impossible, si més no sense malgastar una gran quantitat de diners.

En primer lloc tenim la càmera Kinect dissenyada per Microsoft i llançada al mercat el dia 4 de novembre del 2010 a l'Amèrica del Nord. Aquesta càmera va ser dissenyada exclusivament per fer-se funcionar amb la consola d'última generació Xbox 360 però avui en dia ha resultat factible realitzar altres aplicacions en altres àmbits com és el cas del control de robots mòbils com els Mindstorms de LEGO o fins i tot robots humanoides. L'ús d'aquesta càmera ha estat tant extens que fundacions com ROS[1], una empresa dedicada a oferir Software lliure, ha desenvolupat un software per a desenvolupar aplicacions amb la Kinect. No tan sols això, sinó que el mateix Matlab ofereix una llibreria dedicada per a poder utilitzar les dades que capta aquesta.

Figura 1.1. Càmera Kinect. Extreta de: http://ecx.images-amazon.com/images/I/516DD%2BTj73L._SL1300_.jpg

Per altra banda, un altre gran dispositiu, que per a molts ha pogut passar desapercebut, és el Leap Motion, desenvolupat per Leap Motion Incorporation. Un dispositiu llançat al mercat a finals del 2012 que permet captar dades de les nostres mans i dits. El qual s'està utilitzant actualment per a desenvolupar programes per a plataformes com Windows o el sistema operatiu d'Apple, per tal d'interactuar directament amb aquests mitjançant les mans d'un individu. La Wikipèdia, el defineix com un dispositiu anàleg que substitueix al ratolí dels ordenadors.

[1] ROS: és un conjunt de llibreries "open-source" per tal de desenvolupar software mitjançant robots.

Figura 1.2. Dispositiu Leap Motion. Extreta de: http://i.kinja-img.com/gawker-media/image/upload/sbPDPHpgYc_fit,fl_progressive,q_80,w_636/uxgcfuxdufcgkkyt9oxf.jpg

Finalment, els aparells que s'utilitzaran en aquest projecte per ser controlats seran el drone de Parrot anomenat AR Drone 2.0. i per una altra banda farem ús de la placa Arduino UNO.

Figura 1.3. AR Drone 2.0. Extreta de: http://www.ardronespain.es.

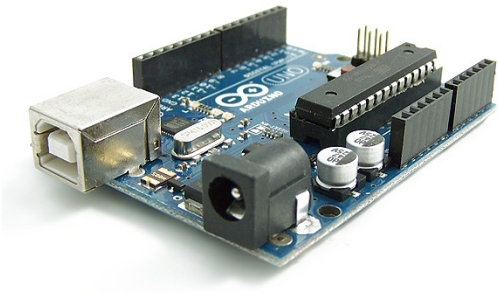

Figura 1.4. Arduino UNO. Extreta de: http://www.cooking-hacks.com/.

2 Objectiu

L'Objectiu d'aquest treball de fi de grau és aconseguir realitzar diferents aplicacions mitjançant la interacció home-màquina. Per tant, fent ús tant de la Kinect com del Leap Motion, es realitzarà inicialment el control d'un drone anomenat AR Drone de Parrot. A més, en el cas del Leap Motion es realitzaran altres aplicacions mitjançant Arduino. D'aquesta manera es té una idea de l'abast d'aquest dispositiu i la gran varietat d'aplicacions que es poden dur a terme. A més per dur a terme el descrit es realitzarà una anàlisi i estudi tant de la Kinect com del Leap Motion per ser utilitzats amb el programa Matlab.

La intenció final d'aquest treball és crear dos programes mitjançant el software Matlab per tal de monitorar les dades tant de la Kinect, del Leap Motion i el AR Drone i a més poder-lo controlar. A més a més, es remarcarà el que es pot realitzar amb Matlab i la Kinect. Així com programes senzills mitjançant Simulink o el codi Matlab per crear gràfiques de les dades que captura aquesta.

Finalment, en cas que s'aconsegueixin els dos objectius principals, és a dir, els programes per controlar el drone mitjançant els dos dispositius, es realitzarà una breu conclusió i comparació de l'experiència i l'ús d'aquests. D'aquesta manera podrem observar que dos dispositius que aparentment són molt diferents però fent ús d'una tecnologia molt semblant poden realitzar, si fa no fa, les mateixes tasques. La comparació, sols es realitzarà amb el cas del control del drone, ja que no s'ha realitzat cap aplicació amb Arduino mitjançant l'ús de la Kinect. A més a més, els resultats seran gravats en vídeo i publicats a la pàgina web de Youtube.

3 Especificacions dels dispositius utilitzats

L'objectiu d'aquest apartat és descriure tots els dispositius que s'han utilitzat per dur a terme amb èxit l'abast d'aquest treball. Així doncs es detallarà el Hardware que contenen els sensors o dispositius d'entrada de dades, és a dir, la càmera Kinect i el Leap Motion; i els actuadors que s'utilitzaran: el AR Drone 2.0 i l'Arduino UNO. A més a més, s'explicarà el software necessari i els passos a seguir per dur a terme la seva instal·lació per tal d'utilitzar-los amb el programa Matlab.

3.1 Característiques de la Kinect

En aquest apartat es descriuran els diferents elements en què està constituïda la Kinect. Recordem que és un element que capta imatges RGB[2] i mitjançant un emissor d'infrarojos i el sensor de profunditat és capaç de captar dades del seu entorn així com les diferents parts del cos d'una persona.

3.1.1 Hardware

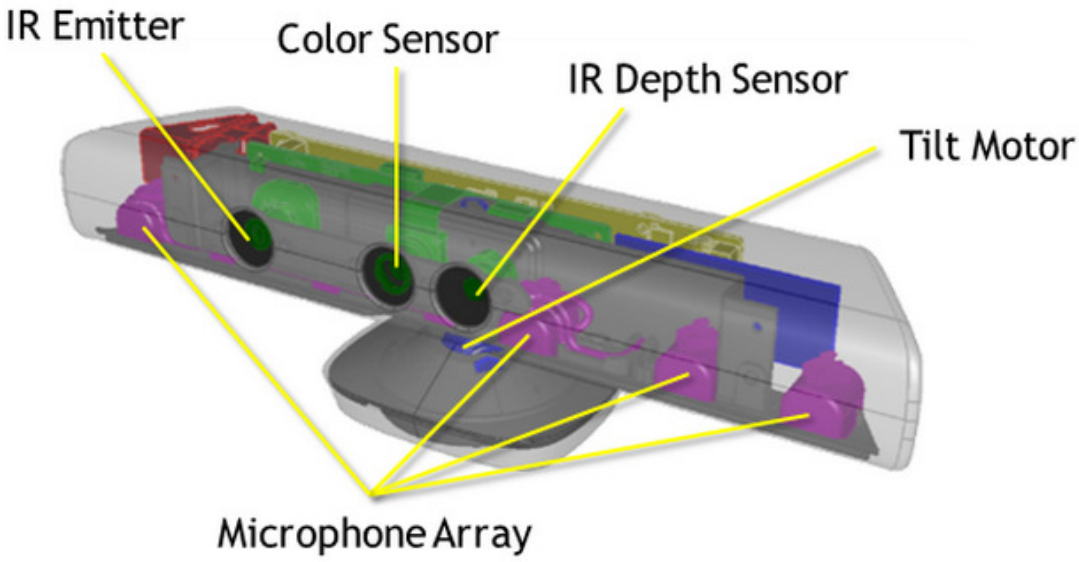

Figura 3.1. Imatge que mostra els diferents components de la Kinect. Extreta de: https://msdn.microsoft.com/en-us/library/jj131033.aspx

La Kinect disposa de diferents sensors i càmeres:
- Un emissor d'infrarojos.
- Un sensor de profunditat d'infrarojos.
- Una càmera RGB d'una resolució de 1280x960 píxels.
- 4 micròfons.
- Un acceleròmetre per als tres eixos.
- Un motor per controlar l'enfocament de la Kinect verticalment ±27°.
- La freqüència de mostreig és de 30 Hz.

3.1.2 Software

En aquest apartat es descriurà e software utilitzat per adquirir la informació d'un cos en moviment que ens proporciona la Kinect:

- Kinect for Windows SDK v1.6.
- Matlab 2014b.
- Microsoft Windows SDK v7.1.
- Microsoft Visual Studio 2010 Express.

[2] RGB: és un model per representar els colors en funció en la composició dels colors primaris: roig, verd i blau.

Per construir el programa per realitzar els càlculs tan sols es farà ús del programa Matlab, la resta de programes tan sols s'utilitzaran perquè el Matlab es pugui comunicar amb la Kinect.

Figura 3.2. Eixos de coordenades de la Kinect. Imatge extreta de: https://hidale.com/wp-content/uploads/2014/08/kinectv1.png.

3.1.3 Adquisició de dades de la Kinect

A través de Matlab podem adquirir dades de la Kinect mitjançant dues metodologies: una és mitjançant comandes Matlab i l'altra utilitzant blocs del Simulink. En aquest cas, s'ha treballat ambdues metodologies.

Inicialment, s'ha estudiat la captació d'informació mitjançant el Simulink, ja que és una interfície gràfica mitjançant blocs molt fàcils d'utilitzar però també és una eina que consumeix molts recursos informàtics i per tant un cop entès el funcionament de la Kinect s'utilitzaran comandes per a comunicar-se amb la Kinect. A continuació descriurem les característiques.

3.1.3.1 Connexió entre la Kinect i Matlab

En aquest apartat es realitzarà una petita explicació per tal d'instal·lar el software necessari per fer funcionar la Kinect amb el Matlab.

Primer de tot s'ha d'instal·lar els compiladors de C++. Es recomana instal·lar el Visual Basic 2010 C++ Express perquè el programa ja inclou totes les llibreries i compiladors necessàries. Un cop instal·lat, s'ha d'instal·lar el Microsoft Windows SDK v7 i Kinect for Windows SDK v1.6.

Finalment, si ja està instal·lat el Matlab, aquest mateix facilita descarregar el suport per Hardware. Vegeu la figura 3.3:

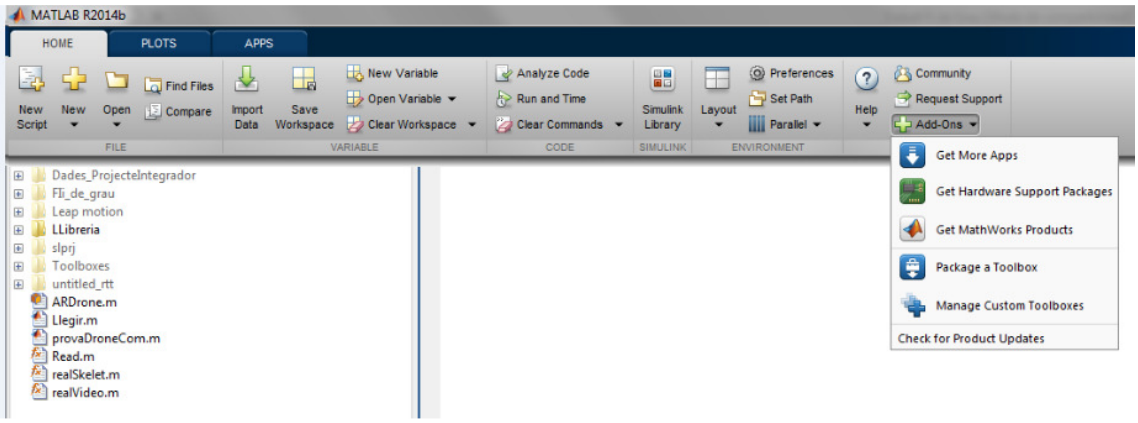

Figura 3.3. Captura de pantalla del Matlab R2014b.

Clicant a "Get Hardware Support Packages" ens apareixerà la següent vinyeta:

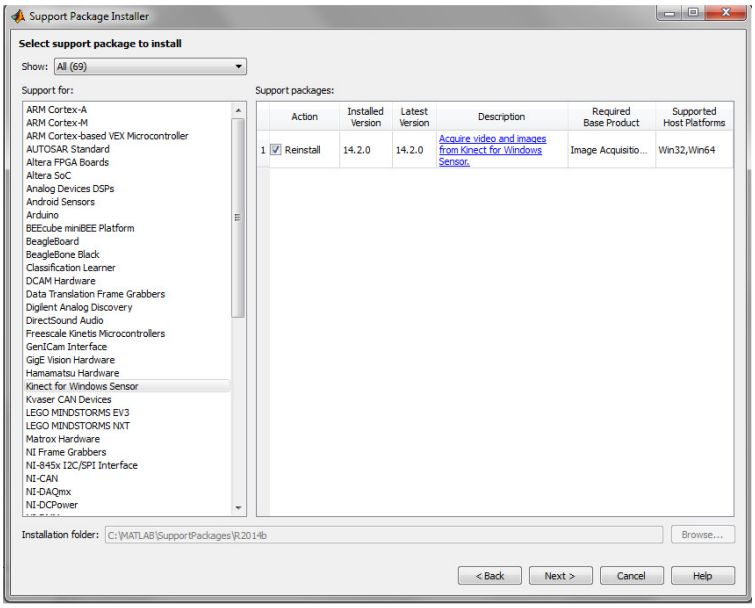

Figura 3.4. Captura de pantalla del "Support Package Installer".

Busquem el Hardware de la Kinect i l'instal·lem. Un petit apunt que m'agradaria destacar pel que fa a la connexió amb el Matlab i Kinect, és els problemes que m'han donat versions anteriors a la 2014b. Per tant, recomano la utilització d'aquesta versió en comptes d'altres.

3.1.3.2 Simulink

El programa Matlab és una eina que permet realitzar càlculs matemàtics, fins a la mateixa creació i simulació de programes. En aquest cas farem ús d'una eina anomenada "Simulink" que ens permet realitzar simulacions mitjançant diagrames de blocs.

Els blocs de Simulink que s'han utilitzat en aquest treball han estat proporcionats d'una llibreria que el mateix Matlab ofereix per internet, anomenada "Simulink suport for Kinect" i que ha estat creada per Takashi Chikamasa. Alguns d'aquests són:

NID IMAQ
Aquest bloc s'encarrega d'adquirir totes les dades de la Kinect. Entre altres coses també podem controlar l'angle de la Kinect.

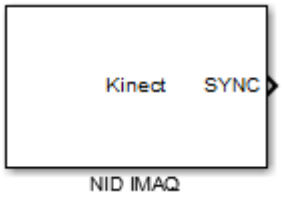

Figura 3.5. Bloc NID IMAQ.

NID Skeleton

Retorna les dades de les articulacions captades per la Kinect.

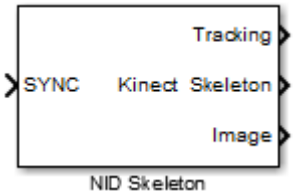

Figura 3.6. Bloc NID Skeleton

És el bloc principal que serà utilitzat per captar dades de l'esquelet. Aquest bloc retorna una matriu de 20 x 3 x 6 de tipus doble. En aquesta matriu es troben els nodes dels esquelets que la Kinect detecta en coordenades reals o projectives x, y i z. En el nostre cas farem ús de les coordenades reals.

Els índexs de cada node de l'esquelet són els següents, aquests estan anomenats en anglès:

Nom	Índex	Nom	Índex
Hip center	1	Wrist right	11
Spine	2	Hand right	12
Shoulder center	3	Hip left	13
Head	4	Knee left	14
Shoulder left	5	Ankle left	15
Elbow left	6	Foot left	16
Wrist left	7	Hip right	17
Hand left	8	Knee right	18
Shoulder right	9	Ankle right	19
Elbow right	10	Foot right	20

Taula 3.1. Enumeració de les unions de l'esquelet.

La imatge següent mostra els nodes representats al famós dibuix "L'Home de Virtruvi":

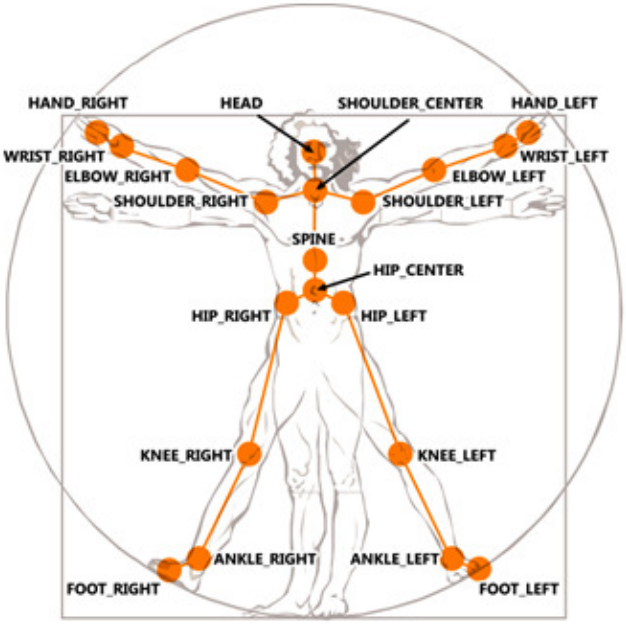

Figura 3.7. Home de Virtruvi.[1]

Finalment, la figura 3.8 mostra com estan distribuïts els eixos de coordenades vistos des del punt de vista de la Kinect.

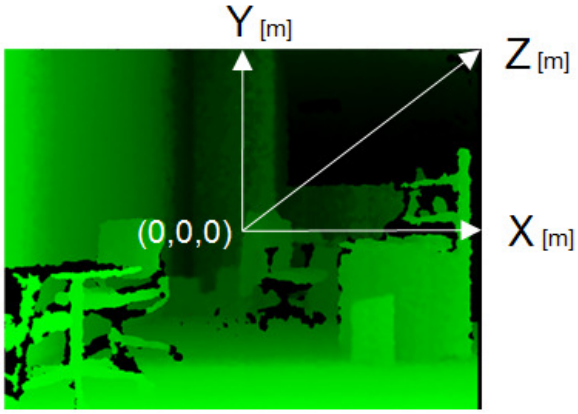

Figura 3.8. Eixos cartesians de la Kinect [1].

Cal destacar que aquesta llibreria inclou exemples per entendre el funcionament de cada un d'aquests blocs. En aquest cas tan sols s'han definit els dos blocs més utilitzats. Per a veure la llista completa que inclou la llibreria vegeu a l'apartat annexes "Blocs Simulink Kinect".

3.1.3.3 Comandes Matlab per la Kinect

En aquest apartat es definirà les comandes principals necessàries per comunicar-se amb la Kinect i obtenir dades com les imatges de vídeo o les dades adquirides pel sensor de profunditat, entre d'altres. Aquest apartat tan sols és perquè el lector es faci una idea de les comandes més utilitzades. Per una millor comprensió es recomana consultar l'apartat d'annexes "Descripció de la llibreria de comandes Kinect".

Algunes de les comandes més destacades que trobarem seran les següents:

videoinput
És la comanda principal la qual ens permet crear un objecte per tal d'utilitzar el sensor de profunditat o la càmera per adquirir imatges.

getselectedsource
Ens permet configurar la Kinect.

trigger
Comença l'adquisició de les dades de la Kinect.

getdata
Ens proporciona les dades adquirides per la Kinect. En cas que utilitzem la càmera, ens retorna les imatges captades i en cas que utilitzem el sensor de profunditat, ens retorna els punts principals de l'esquelet i la imatge de profunditat.

3.2 Característiques del Leap Motion

El Leap Motion és un dispositiu capaç de detectar les dues mans i els 5 dits de cada una d'elles. Per aconseguir-ho, fa ús del següent hardware i software:

Figura 3.9. Imatge que mostra les diferents perspectives del Leap Motion, Imatge extreta de: http://www.laptopmag.com/reviews/accessories/leap-motion-controller.

3.2.1 Hardware

El dispositiu està format per dos càmeres i tres Leds infrarojos, els quals emeten a una longitud d'ona de 850 nanòmetres. Observeu la figura 3.10:

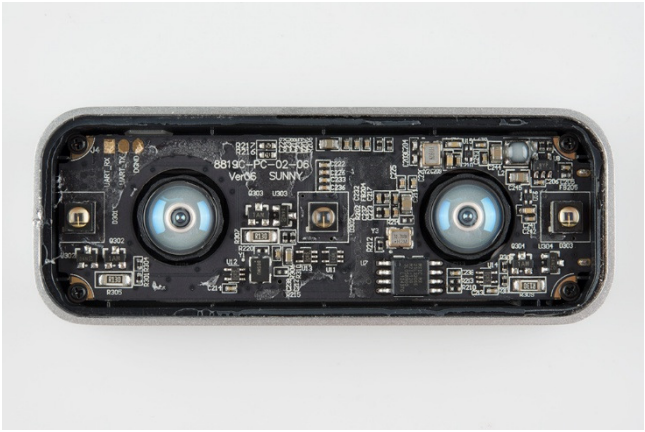

Figura 3.10. Imatge dels components interiors del Leap Motion. Imatge extreta de la pàgina: https://learn.sparkfun.com/tutorials/leap-motion-teardown.

Els Leds infrarojos creen una piràmide invertida els quals creen un volum d'interacció aproximat de 0,227 m^3. A més els Leds arriben a una distància per damunt del dispositiu de 60,96 cm, per 60,96 cm d'amplitud per cada banda i per 60,96 cm de profunditat per cada banda. Observeu la figura 3.11.

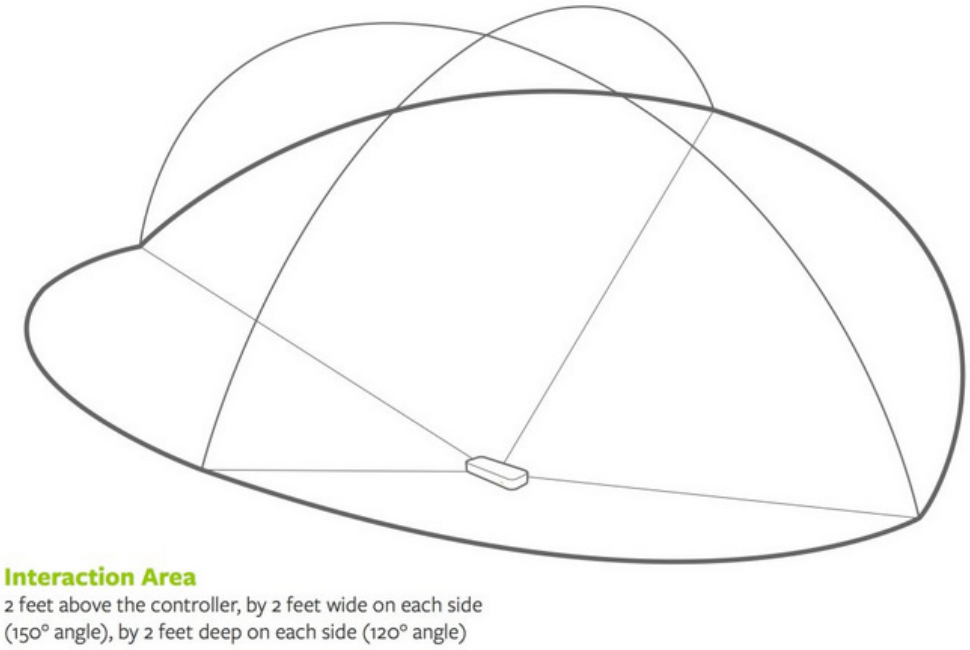

Figura 3.11. Àrea d'interacció del L.M.. Imatge extreta de: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

3.2.2 Software

Igual que en la Kinect, el programa amb què ens comunicarem i agafarem dades del dispositiu Leap Motion serà Matlab.

El software necessari és el següent:
- Matlab 2014b.
- Leap Motion SDK v2.2.5.26752.

Requisits mínims per fer ús del Leap Motion: (Afegir Taula)
- S.O. Windows 7.
- Intel® Core™ i3.
- 2 GB RAM.
- Port USB 2.0.
- Connexió a internet.

3.2.3 Adquisició de dades amb el Leap Motion

Per a l'obtenció de dades del Leap Motion mitjançant el Matlab farem ús d'uns arxius proporcionat per un programador anomenat Jeff Perry el qual ha creat un programa Mex-file[3] que permet obtenir dades dels dits de les mans.

Les dades que proporciona són les següents de cada un dels dits de les nostres mans:

- Posició relativa en coordenades x, y i z.

[3] Mex-file: programes o subrutines escrits en llenguatge C, C++ o Fortran.

- Velocitat del moviment en coordenades x, y i z.
- Direcció i sentit en coordenades x, y i z.
- Variable booleana que retorna si els dits estan estesos o no.

La següent figura mostra la direcció dels eixos de coordenades x, y i z:

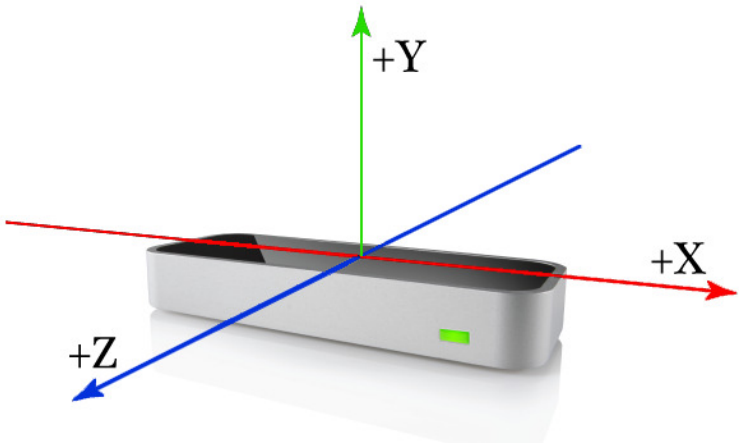

Figura 3.12. Imatge que mostra la direcció dels eixos de coordenades x, y i z. Imatge extreta de: https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Overview.html.

Les distàncies que calcula el Leap Motion estan mesurades en mil·límetres, el temps en microsegons, la velocitat en mil·límetres/segon i els angles en radians.

3.2.3.1 Connexió entre Matlab i Leap Motion

Igual que en el cas de la Kinect, per utilitzar el Leap Motion amb el programa Matlab, haurem d'instal·lar programes seguint els següents passos:

Primer, instal·larem el SDK del dispositiu, és a dir, Leap Motion SDK v2.2.5.26752. A poder ser l'última versió del programa. Un cop instal·lat, descarregarem de la pàgina Github[2] els arxius que hem anomenat a l'apartat anterior. Es descarregarà un "Zip" on trobarem una carpeta anomenada "matleap-master" la qual conté els arxius necessaris per adquirir dades del Leap Motion.

Un cop descarregats els arxius i els obríssim amb el Matlab, a la ruta on estiguin aquests s'ha d'afegir una carpeta anomenada LeapSDK, que la trobarem amb el Software descarregat del Leap Motion SDK. A més, un cop instal·lat aquest últim, anirem a la ruta d'instal·lació i buscarem un arxiu anomenat "Leap.dll" i el copiarem a la ruta dels arxius de la carpeta "matleap-master". Finalment, la carpeta ha de contenir els següents arxius que veureu a la figura 3.13.

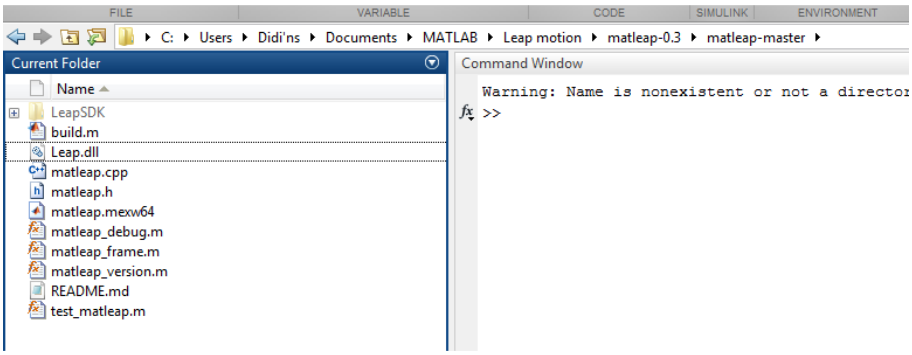

Figura 3.13. Captura de pantalla del contingut de la carpeta "matleap-master".

El codi de l'arxiu que s'encarrega de proporcionar-nos els fotogrames de les dades el podeu consultar a l'apartat annexes: "Arxiu Mex per l'obtenció de dades del Leap Motion".

3.3 Característiques del Drone Parrot

En aquest apartat es descriurà les característiques principals del AR Drone 2.0 de la companyia francesa Parrot. Un dispositiu volador de quatre hèlices RC, que és controlat normalment amb Smartphones gràcies al fet que disposa d'una CPU i sensors. Gràcies a les dues càmeres que disposa, permet captar imatges i vídeos del nostre entorn.

Figura 3.14. Dídac Coll, fotografia del AR Drone de Parrot, 20 de maig del 2015.

La seva primera versió, el AR Drone 1.0. va ser revelat per primera vegada al "International CES[4]" de 2010.

3.3.1 Hardware

Electrònica

- Càmera HD. 720 p a 30 FPS.
- Processador d'1 GHz i 32bit ARM Cortex A8.
- RAM DDR2 d'1 GB a 200 MHz.
- USB 2.0 d'alta velocitat per extensions.
- Wi-Fi.
- Giroscopi de 3 eixos amb una precisió de 2000 º/s.
- Acceleròmetre de 3 eixos amb una precisió de +/- 50 mg.
- Magnetòmetre de 3 eixos amb una precisió de 6º.
- Sensor de pressió amb una precisió de +/- 10 Pa.
- Sensors d'ultrasons per a mesurar l'altitud.
- Càmera vertical QVGA de 60 FPS per a mesurar la velocitat d'avançament.

Motors

- Motors sense escombretes
- Potència: 14,5 W.
- Velocitat angular: 28.500 RPM.
- Engranatges de Niló per a una relació de transmissió de 8,75.

[4] International CES: de l'anglès (Internacional Consumer Electronics Show), és una fira internacional de consum electrònic que es duu a terme cada any a les Vegues.

Dimensions

- **Amb coberta protectora**

  - Llargada 52,5 cm
  - Amplada 51,5 cm
  - Pes 380 g(exterior) i 420 g(interior)

- **Sense coberta protectora**

  - Llargada 45 cm
  - Amplada 29 cm
  - Pes 340 g

3.3.2 Software

La versió del Software que incorpora és: Linux 2.6.32.
El Software per a treballar des de l'ordenador amb el drone és Matlab 2014b.

3.3.3 Connexió entre el Matlab i el AR Drone

La connexió entre el drone i l'ordenador es realitzarà mitjançant "Wi-Fi". A més farem ús d'un arxiu extret de MathWorks[4]. L'arxiu inclou tot el necessari per establir connexió entre el drone i Matlab. De totes maneres l'arxiu està una mica incomplet i dóna errors i la comunicació que estableix entre el Hardware i el Software no és precisament l'esperada. Per aquest motiu més endavant es comentarà les modificacions que s'han cregut adients perquè s'adapti a les nostres necessitats.

Un cop explicat l'anterior, procedirem a descriure en què consisteix l'arxiu el qual el podeu trobar complet a l'apartat d'annexes "Classe que mostra les funcions bàsiques del AR Drone". Aquest és una classe el qual conté una sèrie de funcions i una petita GUI. Aquesta última permet controlar el drone amb les tecles de l'ordenador i mostra per pantalla les dades de navegació del dispositiu.

Si observem el codi de l'arxiu, la connexió s'estableix mitjançant la comanda "udp". Vegeu el codi següent:

```
obj.ARc = udp('192.168.1.1', 5556, 'LocalPort', 5556);
open(obj.ARc);
```

Aquest cas s'encarrega d'obrir el port per on s'envien les comandes de moviment del drone. El següent codi mostra com obrir el port de comunicacions per rebre dades de navegació del drone:

```
obj.ARn = udp('192.168.1.1', 5554, 'LocalPort', 5554, ...
              'ByteOrder', 'littleEndian', ...
              'InputBufferSize', 500, ...
              'BytesAvailableFcn', {@navPacketRxCallback,obj}, ...
              'DatagramTerminateMode', 'on', ...
              'BytesAvailableFcnMode', 'byte', ...
              'BytesAvailableFcnCount', 24);
fopen(obj.ARn);
% Byte to NavData port for initialization
 fwrite(obj.ARn, 1);
 % NavData command to command port
 AR_NAV_CONFIG = ...
```

```
        sprintf('AT*CONFIG=2,\"general:navdata_demo\",\"TRUE\"\r');
fprintf(obj.ARc, AR_NAV_CONFIG);
```

Inicialment, estableix un enllaç udp amb el port 5554 i la IP 192.168.1.1. Configura el tamany d'entrada del bus perquè com màxim emmagatzemi 500 B. A més hi ha creada una funció que és cridada quan hi ha bytes disponibles.
Finalment, perquè el drone comenci a enviar dades, s'utilitza la comanda "fwrite" per enviar-li un byte i aquest respongui enviant les dades.

3.4 Característiques d'Arduino UNO

La placa Arduino UNO, és un dispositiu electrònic reprogramable de molt baix cost econòmic, emprat per a desenvolupar aplicacions simples relacionades l'electrònica. Aquesta disposa d'entrades i sortides digitals i també d'entrades i sortides analògiques. Això permet desenvolupar una gran varietat d'aplicacions tan simples com complexes. Des d'encendre i apagar un LED a construir un robot.

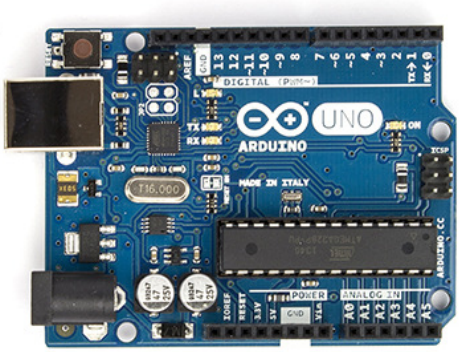

Figura 3.15. Cara davantera de l'Arduino UNO, extreta de: http://www.arduino.cc/en/Main/ArduinoBoardUno.

A diferència d'altres plaques, aquesta ha estat preprogramada mitjançant un "Bootloader[5]", això permet que aquesta placa no hagi de fer ús d'un programador extern i per tant, tan sols faci falta un simple ordenador per programar-la. A continuació, descriurem el Hardware i el Software que farem ús:

3.4.1 Hardware

Electrònica

- Microcontrolador	ATmega328
- Voltatge d'operació	5 V
- Voltatge d'entrada (recomanat)	7-12 V
- Voltatge d'entrada (límit)	6-20 V
- Pins E/S Digitals	14 (6 dels quals són sortides PWM)
- Pins E Analògics	6
- Corrent CC per a pins de E/S	40 mA
- Corrent CC per a pins de 3,3 V	50 mA
- Memòria Flash	32 kB, 0,5 kB els ocupa el Bootloader
- SRAM	2 kB
- EEPROM	1 kB
- Freqüència de rellotge	16 MHz

Dimensions

- Llargada	68,6 mm
- Amplitud	53,4 mm
- Pes	25 g

[5] Bootloader (Gestor d'Arranc): és un programa senzill que no té la totalitat de les funcionalitats d'un sistema operatiu, i que està dissenyat exclusivament per preparar tot el que necessita el sistema operatiu per funcionar.

3.4.2 Software

- Arduino 1.6.4
- Matlab 2014b

3.4.3 Connexió entre Matlab i Arduino

Per tal de controlar l'Arduino mitjançant Matlab, es requereix instal·lar unes llibreries. Per tant, l'objectiu d'aquest punt és descriure tots els passos necessaris per fer funcionar-la amb Matlab.

Inicialment, s'ha de descarregar del següent enllaç tots els arxius necessaris: http://es.mathworks.com/matlabcentral/fileexchange/32374-matlab-support-for-arduino--aka-arduinoio-package-[5]. A més a més, s'ha d'obtenir el Software oficial d'Arduino de la pàgina Web oficial: http://www.arduino.cc/en/Main/Software[6].

Una vegada instal·lat el software oficial, procedim a obrir i a descomprimir l'arxiu Zip que hem descarregat de la pàgina web de Matlab. Els arxius que ens han d'aparèixer són els següents que es mostren a la figura 3.16.

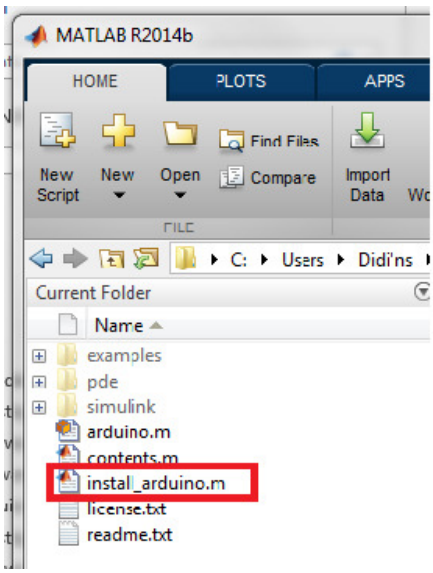

Figura 3.16. Captura de pantalla dels arxius descarregats per fer funcionar Arduino.

Ara tan sols cal obrir l'arxiu "install_arduino.m" i executar-lo. D'aquesta manera instal·lem totes les llibreries necessàries a Matlab.

Finalment, dintre la carpeta "pde" trobarem uns arxius. De tots aquests l'arxiu "adios.pde" l'hem de carregar a la placa Arduino UNO. Per tant, la connectem i obrim el programa oficial d'Arduino per tal de procedir a carregar l'arxiu a la placa. Una vegada realitzat tot el descrit anteriorment, ja podem fer ús de la placa mitjançant Matlab. Per a més informació sobre les comandes, es recomana consultar l'apartat d'annexes: "Comandes Matlab per a Arduino".

4 Kinect

Un cop s'ha tingut clar l'objectiu del treball i plantejats els fonaments teòrics, s'ha procedit a realitzar proves amb la Kinect. Per comprendre millor el funcionament d'aquesta inicialment, s'ha fet mitjançant el Simulink. Aquest ha permès entendre millor com treballa la Kinect i ha facilitat molt la feina alhora de desenvolupar el programa final. Algunes de les coses més destacades que ens ha permès els digrames de blocs, és obtenir imatges RGB, dades de l'esquelet, imatges de profunditat, etc.

Es poden consultar els resultats del control del drone mitjançant la càmera a l'apartat annexes: "Enllaç del vídeo de la Kinect".

4.1 Proves realitzades amb Simulink

Mitjançant les dades obtingudes de l'esquelet, s'ha realitzat un diagrama de blocs que ha servit per calcular els angles de l'esquelet d'un cos d'una persona.

4.1.1 Càlcul d'angles

Aquest exemple que il·lustrarem a continuació, calcula els angles dels dos: colzes, espatlles i fins i tot l'angle guinyada de les espatlles. El programa ha estat implementat mitjançant una GUI. Observem la figura 5.1.

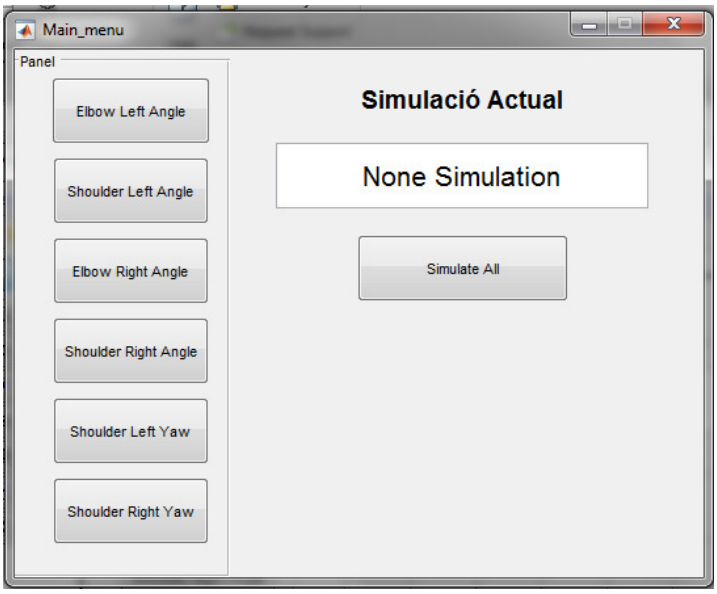

Figura 4.1. Programa que mostra els angles dels braços.

Aquest petit programa realitzat amb la GUI de Matlab permet mostrar en gràfics cada un dels angles anomenats anteriorment. En aquest cas prendrem de referència el que succeeix quan polsem el botó amb l'etiqueta "Elbow Left Angle". Un cop el polsem, se'ns mostrarà els dos gràfics que mostren les dades que captura la Kinect del angle del colze esquerre. Un mostra les dades que adquireix i l'altra les fem passar per un filtre el qual més endavant serà descrit.

Tant l'eix Y com el X de les figures 4.2 i 4.3, representen l'angle en **graus** i el **temps** en segons, respectivament.

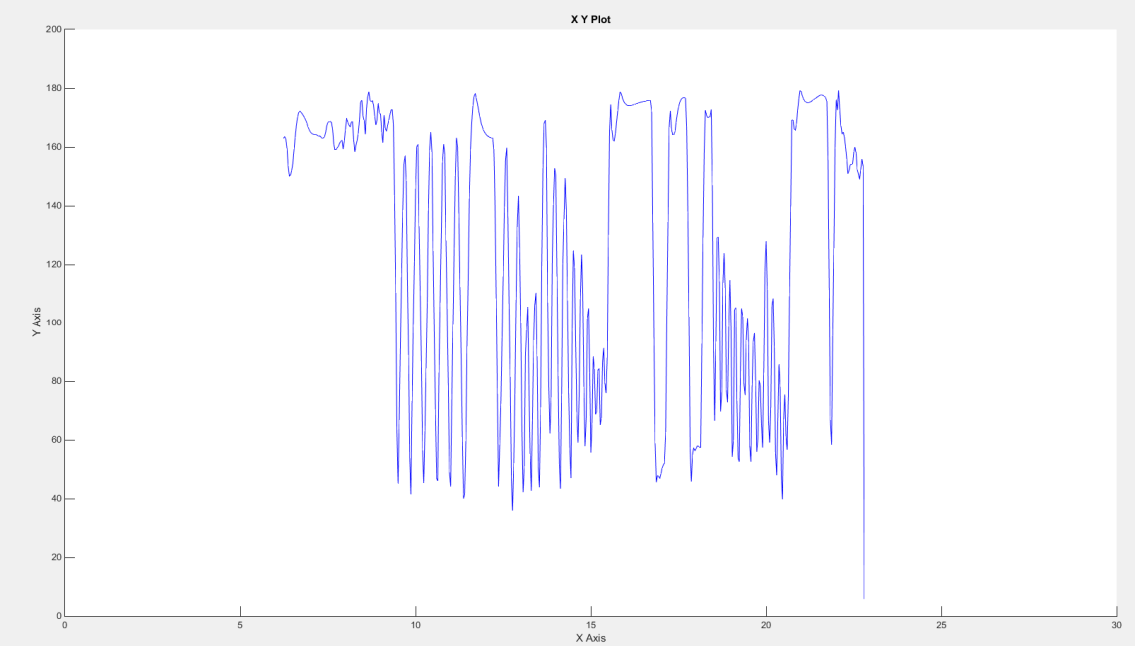

Figura 4.2. Dades de l'angle del colze esquerre. Eix Y representa els angles en graus i l'eix X representa el temps en segons.

En aquesta primera prova ha consistit a obrir i tancar ràpidament el colze, observem a la figura 4.2 que si les dades no les passem per un filtre, obtenim els màxims pics des d'uns 20º que interpretem que el colze està tancat i uns 170º que és aproximadament obert. Si tenim en compte que aquestes dades han de servir per realitzar un control d'un aparell, en cas que realitzem un control tot o res com per exemple encendre o apagar un LED no ens importa el senyal. Però si per exemple, volem controlar el PWM d'un motor de corrent continu, no podem permetre que rebi un senyal com l'anterior a no ser que el temps de resposta d'aquest sigui molt més gran que les dades que rebem, el qual no és el nostre cas, ja que el temps de resposta del drone és 0,033 s, és a dir, el mateix que la freqüència que rebem les dades de la Kinect.

Per tant, haurem de filtrar el senyal. Com que aquest cas tan sols ha estat una prova, tan sols s'ha fet una mitja per filtrar-la. Vegeu la figura 4.3.

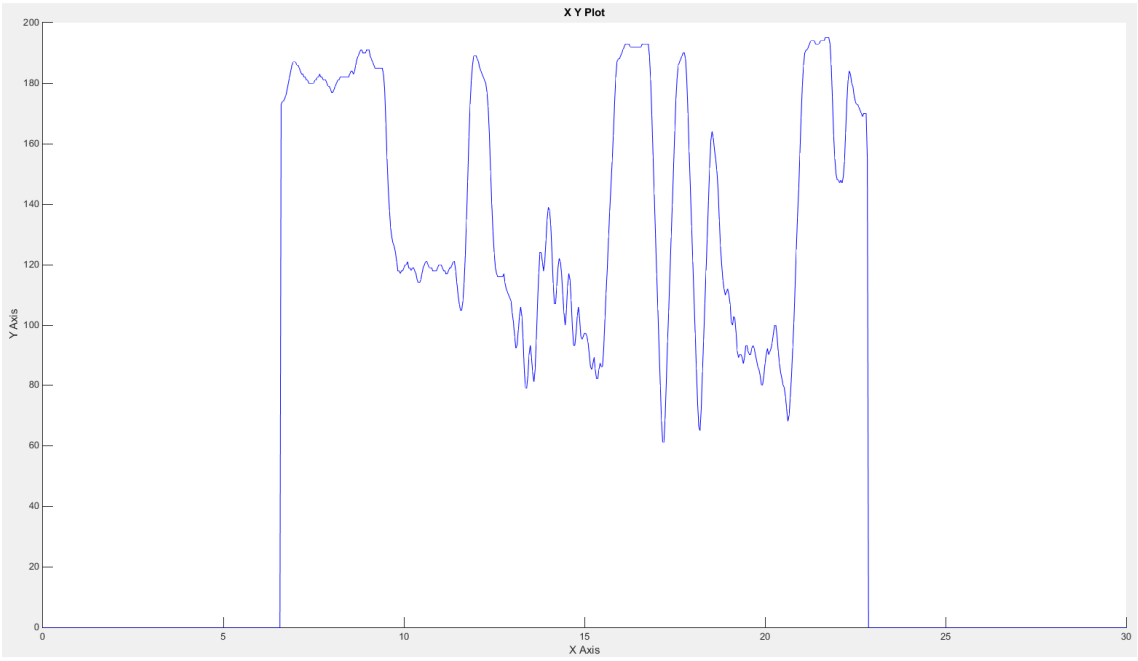

Figura 4.3. Dades de l'angle del colze esquerre filtrades.

Vegeu el contingut del codi de Simulink a la figura 4.4.

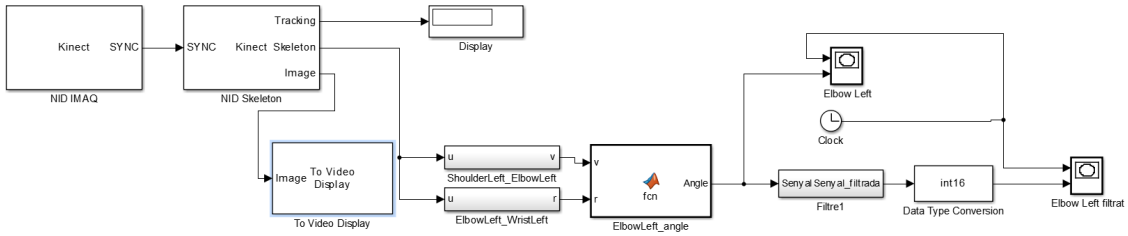

Figura 4.4. Blocs Simulink per calcular l'angle del colze esquerre.

Podem observar que hi ha diversos blocs, dos dels quals estan referenciats a apartats descrits anteriorment. El bloc "To vídeo display" s'encarrega de mostrar la imatge que captura la Kinect a través del sensor de profunditat. Vegeu la figura 4.5.

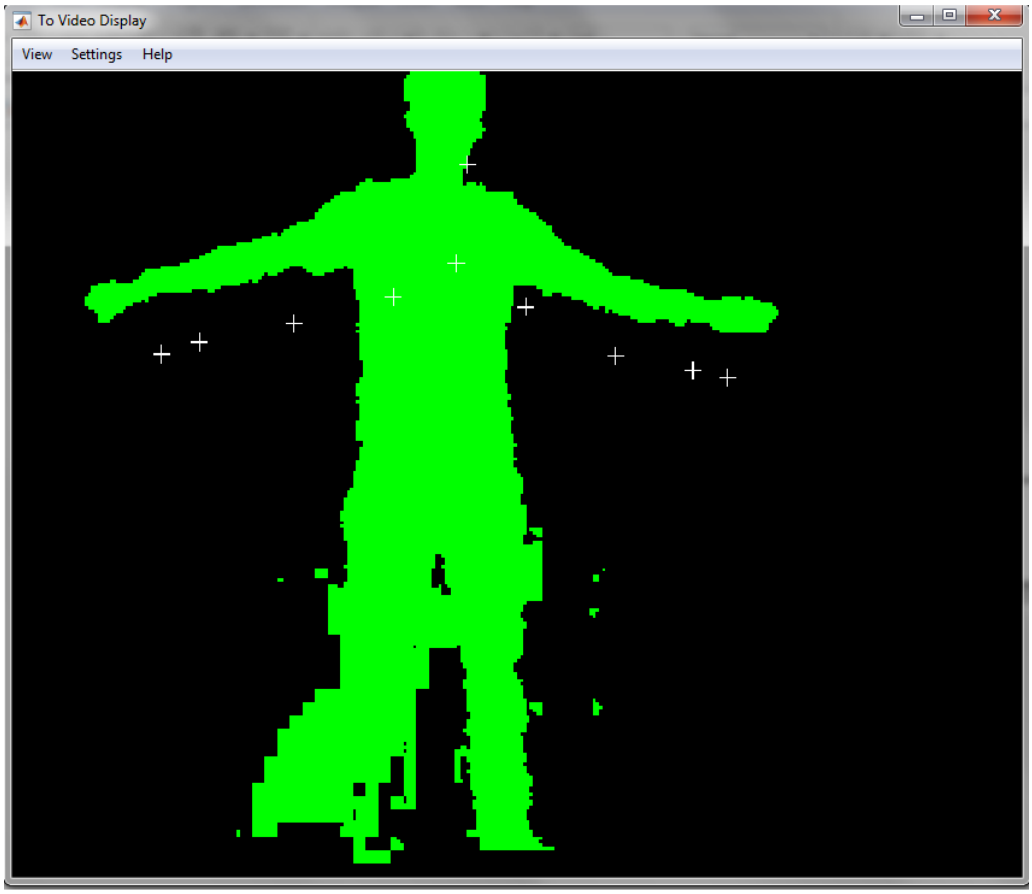

Figura 4.5. Captura de pantalla de les imatges de vídeo que mostra el sensor de profunditat.

Per una altra banda, tenim els blocs que mostrarem a continuació:

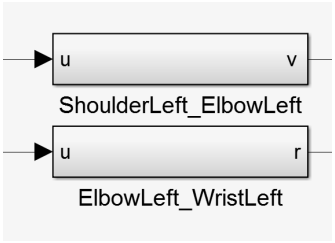

Figura 4.6. Blocs per calcular vectors.

Els dos blocs representats a la figura 4.6 s'utilitzen per calcular el vector entre els nodes de l'espatlla i el colze esquerre; i per calcular el vector entre el colze i el canell esquerre. Recordem que ens arriba una matriu de 20x3x6 per les entrades "u" dels dos blocs. La matriu conté les dades dels sis possibles esquelets que pot detectar la Kinect però en aquest cas tan sols ens interessa la d'un esquelet. Observeu el codi que contenen els blocs anteriors.

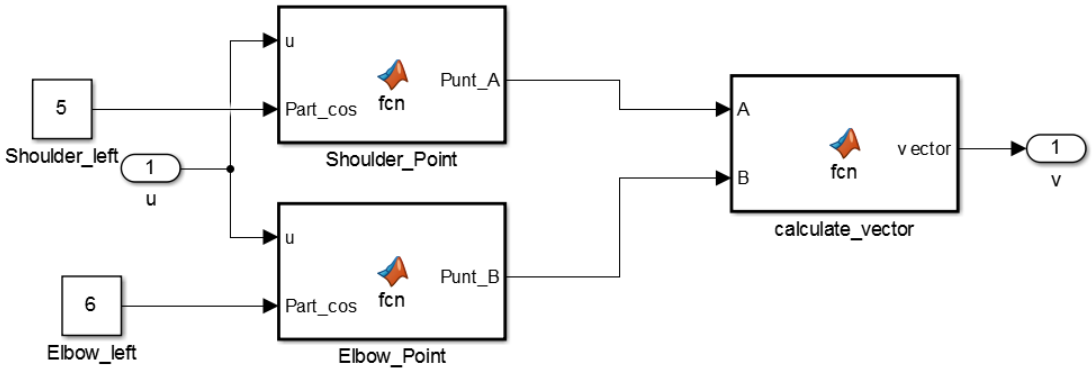

Figura 4.7. Contingut del bloc "ShoulderLeft_ElbowLeft".

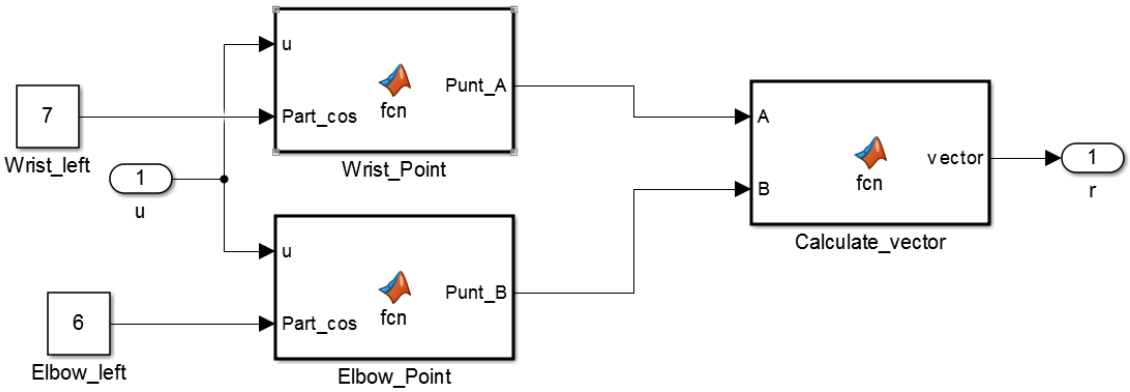

Figura 4.8. Contingut del bloc "ElbowLeft_WristLeft".

El codi consisteix a agafar les dades que ens interessen de la matriu, i les separem en coordenades x, y i z. Això se n'encarreguen els blocs: Shoulder_Point, Elbow_point, etc; mitjançant el següent codi:

```
function Punt_A  = fcn(u,Part_cos)
%#eml

x=u(:,Part_cos,1);
y=u(:,Part_cos,2);
z=u(:,Part_cos,3);

Punt_A = [x y z];
```

Observem que ens arriba la matriu mitjançant un valor assignat a la variable Part_cos. Mitjançant aquesta seleccionem el node que ens interessa. Un cop seleccionat, agafem les components del node en les variables x, y i z i les afegim al vector Punt_A.

Tots els blocs anteriors realitzen els mateixos càlculs, tan sols canvia la part del cos seleccionada. També si ens fixem al valor, per seleccionar l'esquelet posem ':'. El motiu d'això senzillament és per seleccionar les dades del primer esquelet que detecti, ja que l'ús del programa s'ha considerat perquè l'utilitzi tan sols una persona alhora. Per tant, no ens importa quin dels sis esquelets detecti.

Un cop s'ha obtingut el punt del colze i l'espatlla esquerra, la funció "calculate_vector" s'encarrega de calcular el vector entre el colze i l'espatlla.

```
function vector  = fcn(A,B)
%#eml
vector = B - A;
```

S'aplica el mateix procediment per calcular el vector entre el colze i el canell esquerre. Fem ús del bloc "ElbowLeft_angle" i calculem l'angle. Observem el codi següent:

```
function Angle  = fcn(v,r)
%#eml
Producte_escalar = dot(v,r); %Calcul producte escalar
Angle_radians= acos(Producte_escalar/(norm(v,2)*norm(r,2)));
Angle = Angle_radians*180/ pi;
```

Per calcular l'angle fem ús del producte escalar de dos vectors el qual permet calcular el cosinus entre dos vectors. La fórmula matemàtica és la següent:

$$\alpha = \cos^{-1}\frac{A \cdot B}{\|A\| \cdot \|B\|} \qquad (1)$$

On:

- *α: És l'angle més petit entre els dos vectors.*
- *A: és un vector director.*
- *B: és un vector director.*

Mitjançant el bloc anomenat "Filtre 1", filtrem la senyal. Com ja s'havia anomenat anteriorment, el filtre consisteix a realitzar una mitjana de deu dades. La figura 4.9 mostra el contingut del bloc del filtre.

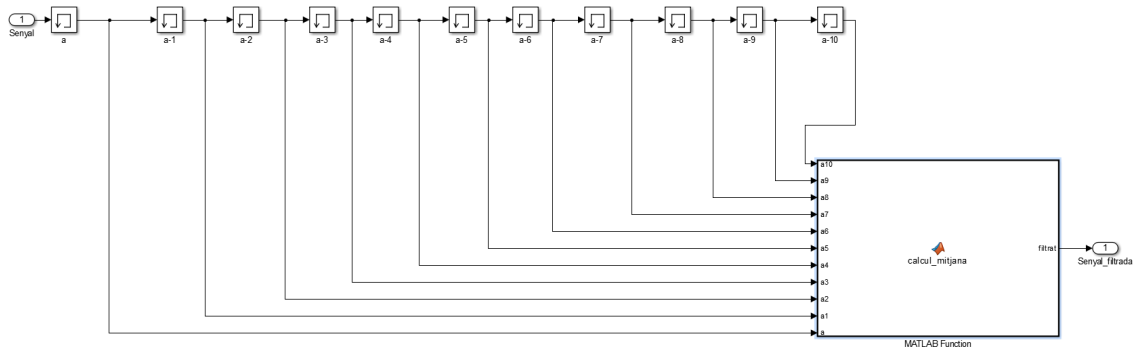

Figura 4.9. Contingut del bloc "Filtre 1".

Finalment, convertim la variable de sortida del filtre en un enter i la mostrem en un gràfic en temps de simulació[6].

El programa no tan sols permet visualitzar cada un dels angles amb gràfics, sinó que també permet visualitzar-los tots alhora mitjançant sis blocs anomenats "Display". Tan sols cal clicar al botó "Simulate all" i el programa obrirà un Simulink permetent la visualització d'aquests. Vegeu la figura 4.10.

[6] Temps de simulació: ens referim al temps que s'està simulant i obtenim les dades de la Kinect mitjançant SImulink.

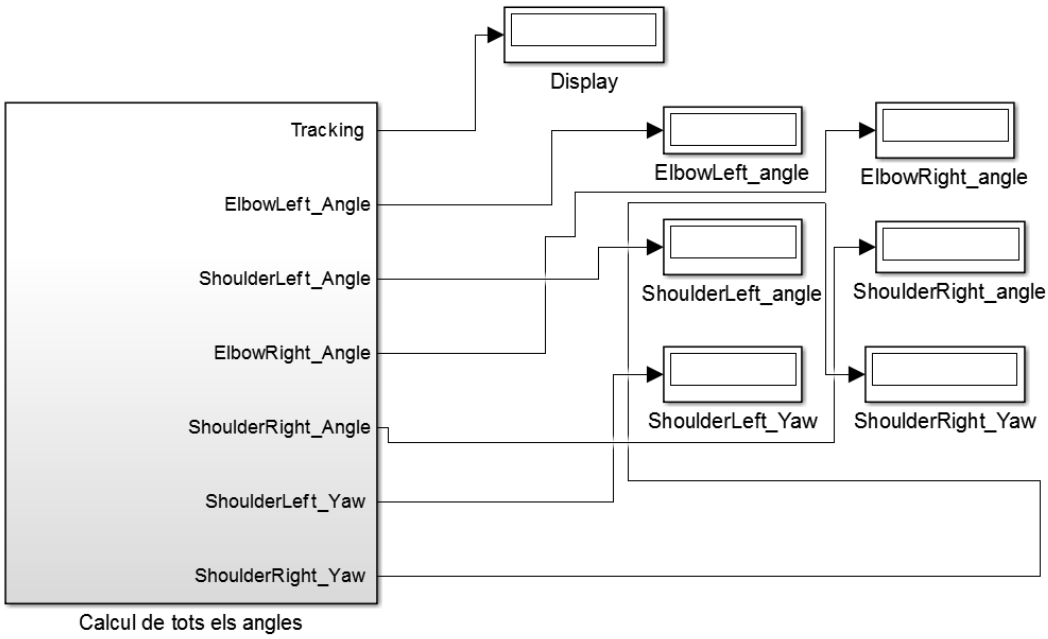

Figura 4.10. Codi Simulink que mostra tots els angles de la part superior del cos.

El codi implementat per calcular cada un dels angles en aquest cas és el mateix que el descrit anteriorment.

4.2 Primeres proves amb comandes

Finalitzades les primeres proves en Simulink, s'ha començat a pensar amb el programa final. La millor opció fins ara ha estat dissenyar la interfície d'aquest amb GUI. Però abans, s'ha hagut de realitzar una sèrie de proves per tal de comprovar que utilitzant tan sols comandes de Matlab era factible calcular els angles. Per tant, abans de realitzar el programa final, s'han anat fent prototips de programes.

El càlcul d'angles es farà de la mateixa manera que s'ha fet amb el Simulink, és a dir, mitjançant la fórmula del producte escalar de dos vectors, ja que podem obtenir el cosinus de l'angle més petit que formen dos vectors.

4.2.1 Mostrar imatges per pantalla

Inicialment, s'ha creat un petit Script per tal de mostrar les imatges que capten la càmera RGB i el sensor de profunditat per tal de familiaritza'ns amb les comandes principals de la Kinect.

```
preview(rgbVideo); % Mostra les imatges RGB
preview(depthVideo); % Mostra les imatges del sensor de profunditat
```

La resta de codi es troba a l'apartat d'annexes: "Script per mostrar imatges per pantalla".

4.2.2 Capturar dades de l'esquelet

El següent exemple mostra com obtenir dades de l'esquelet, és molt semblant a l'altre descrit anteriorment però en aquest cas no mostrarem les imatges que captura el sensor.

```
start(depthVideo);
trigger(depthVideo); %Begin to take data
% Retrieve the frames and check if any Skeletons are tracked
[frameDataDepth, timeDataDepth, DataDepth] = getdata(depthVideo);
stop(depthVideo);
```

El programa anterior, captura 100 fotogrames de dades. Vegeu la resta de codi a l'apartat annexes: "Script per a capturar dades de l'esquelet".

4.2.3 Mostrar l'esquelet en un gràfic 2D

Com molt bé diu el títol d'aquest subapartat, aquesta prova consisteix a mostrar en un gràfic de dues dimensions x i y en temps real les dades que adquireix la Kinect. Per a mostrar l'esquelet es farà ús de la comanda plot, la qual permet mostrar les unions de l'esquelet. El bucle següent mostra les unions de l'esquelet captades per la Kinect fins que nosaltres tanquem la finestra del "Script".

```
while ishandle(f)
    trigger(depthVideo); %Begin to take data
    %Take Frames
    [frameDataDepth, timeDataDepth, DataDepth] = getdata(depthVideo);
    SkeletonTracked = DataDepth.IsSkeletonTracked;
    whatSkeletonIsTracked = find(DataDepth.IsSkeletonTracked);
    if sum(SkeletonTracked)>0   %Check if one of the six skeletons
are...
        ...tracked
%Plot the dataDepth in real coordinates.
```

```
plot(DataDepth.JointWorldCoordinates(:,1,whatSkeletonIsTracked),...
  DataDepth.JointWorldCoordinates(:,2,whatSkeletonIsTracked),'+'),...
        axis([-1 1 -1 1]);
        xlabel('X(m)');
        ylabel('Y(m)');
        grid on;
        drawnow; % upload data
    end
end
stop(depthVideo);
```

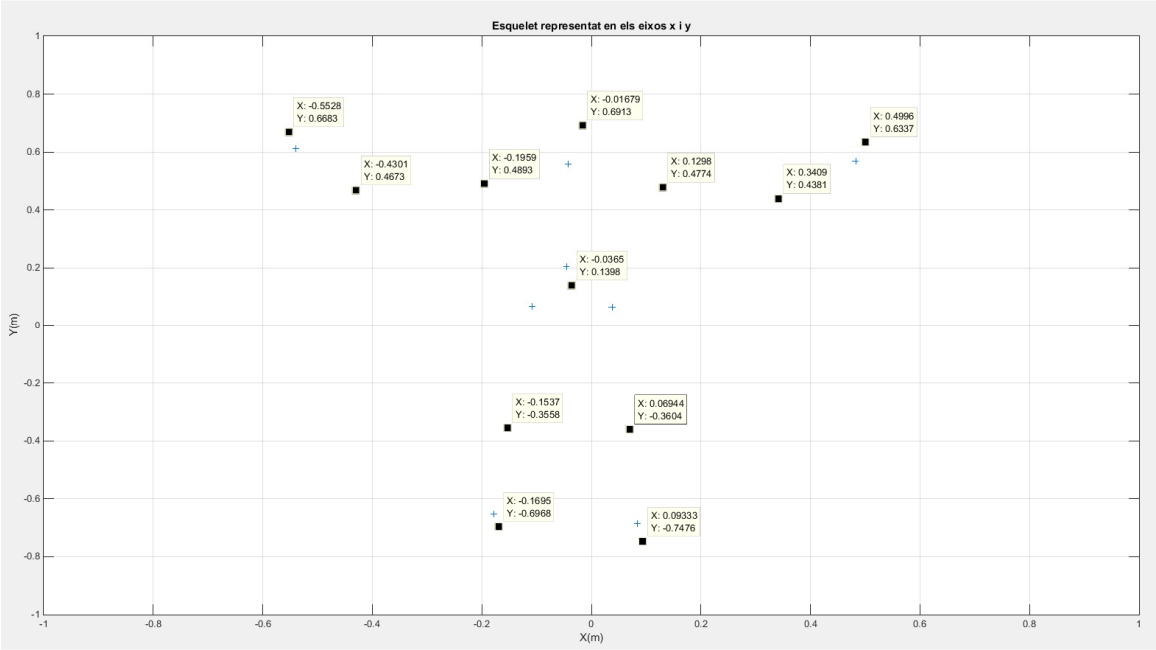

Figura 4.11. Gràfic de l'esquelet representat en els eixos x i y.

Per consultar la resta del codi, vegeu a l'apartat annexes: "Script per a mostrar l'esquelet en 2D".

4.2.4 Mostrar l'esquelet en un gràfic 3D

Molt semblant al cas anterior però en aquest cas el mostrarem l'esquelet en un gràfic en 3D x, y i z. Molt semblant a l'anterior però canviant tan sols el "plot" per "plot3":

```
plot3(DataDepth.JointWorldCoordinates(:,1,whatSkeletonIsTracked),...
  DataDepth.JointWorldCoordinates(:,3,whatSkeletonIsTracked),...
     DataDepth.JointWorldCoordinates(:,2,whatSkeletonIsTracked),...
        '+r','LineWidth',2),axis([-2 2 -0.5 3 -1 1]);
```

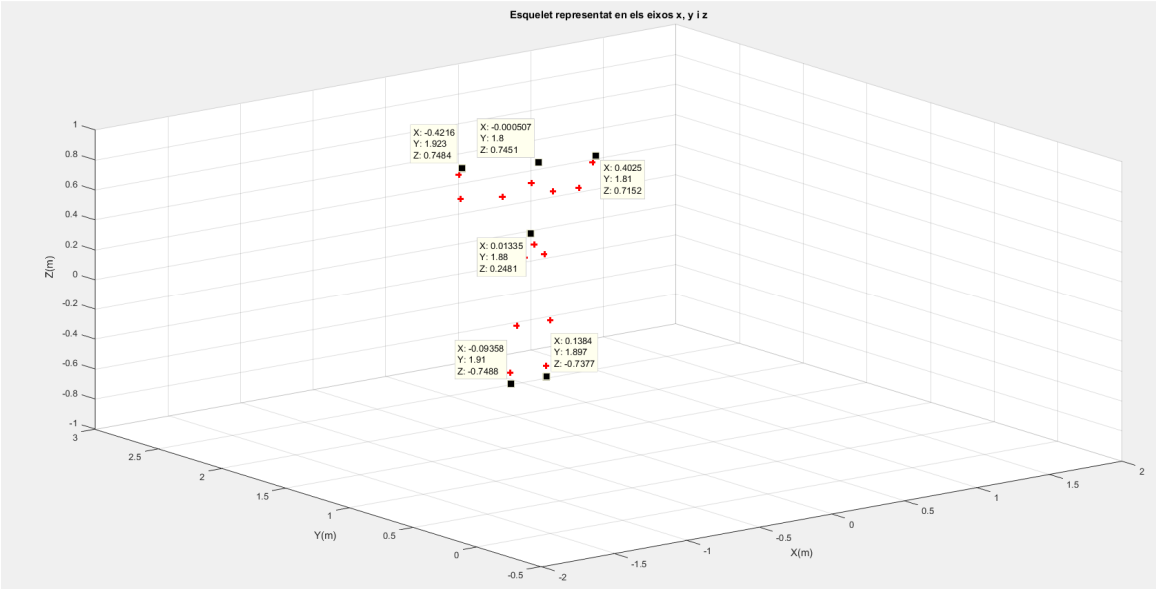

Figura 4.12. Representació dels nodes de l'esquelet captats per la Kinect en els eixos x, y i z

Vegeu la resta de codi a l'apartat annexes: "Script per a mostrar l'esquelet en un gràfic 3D".

4.3 Desenvolupament del programa final

L'objectiu d'aquest apartat és explicar en què consisteix el programa final que s'ha realitzat amb la Kinect per tal de controlar el drone. Aquest apartat estirà dividit en tres subapartats on s'explicarà el següent:

- El desenvolupament de la GUI.
- Funcions utilitzades i creades.
- Connexió Matlab amb el drone.

4.3.1 Desenvolupament de la GUI

Els paràmetres de disseny de la interfície gràfica que s'han tingut en compte han estat els següents:

- **Temps d'execució:** el qual és un dels més importants, ja que estem obtenint dades i controlant un dispositiu en temps real. Recordem que la Kinect adquireix dades cada 0,033 s i el temps mínim en què el drone adquireix una comanda també és el mateix.

- **Informació necessària per a l'usuari:** un aspecte important a tenir en compte però que no ha d'afectar el rendiment del programa.

Per tant, la GUI dissenyada mostrarà la suficient informació a l'usuari: com els angles dels braços i colzes, els fotogrames per segon, etc. Per aquest motiu la interfície serà senzilla sense consumir gaires recursos gràfics.

4.3.2 Interfície gràfica

Aquest apartat té l'objectiu de descriure l'aspecte gràfic del programa que s'ha creat. La interfície gràfica com ja s'ha anomenat anteriorment, s'ha creat per tal d'aconseguir la resposta més ràpida possible entre la càmera Kinect i el drone. Observeu la figura següent:

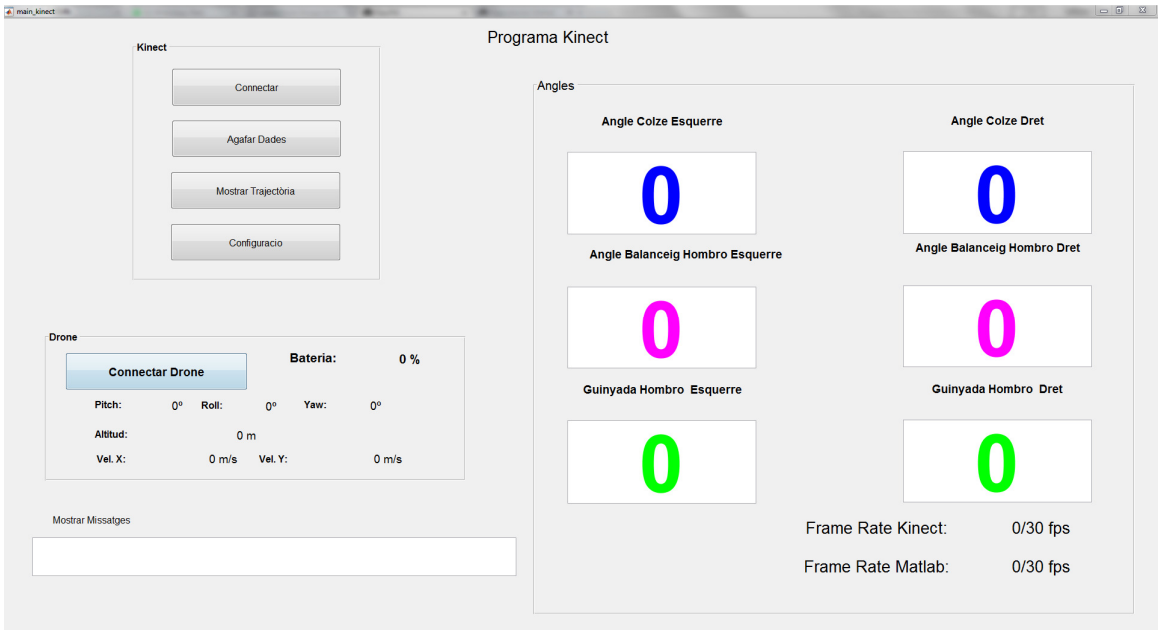

Figura 4.13. Interfície gràfica per a monitoritzar els angles del cos.

Com es pot observar, el programa disposa d'un panell de botons; set editors de text; per on sis d'ells es mostren cada un dels angles calculats i un setè per a mostrar missatges d'errors. A més, el programa també mostra la quantitat de fotogrames per segon tant per les dades que capta el sensor, com les dades que calcula el Matlab.

El programa també disposa d'un panell per monitorar els sensors del drone, així com l'estat de la bateria com la posició d'aquest mitjançant els angles: pitch, roll i yaw.

Panell de botons de la Kinect
Observem quatre botons: connectar, Agafar Dades, plot esquelet i configuració.

- **Connectar:** s'encarrega d'establir la connexió entre la Kinect i Matlab, creant un objecte tipus vídeo per capturar les dades del sensor de profunditat.

- **Agafar Dades:** s'encarrega de començar agafar dades i controlar el drone mitjançant les dades que s'estan recollint.

- **Mostrar Trajectòria:** Aquest s'encarrega de mostra'ns la trajectòria del drone que aquest ha realitzat durant el control.

- **Configuració:** configura els paràmetres de la Kinect obrint la següent finestra que es pot observar a la figura 4.14.

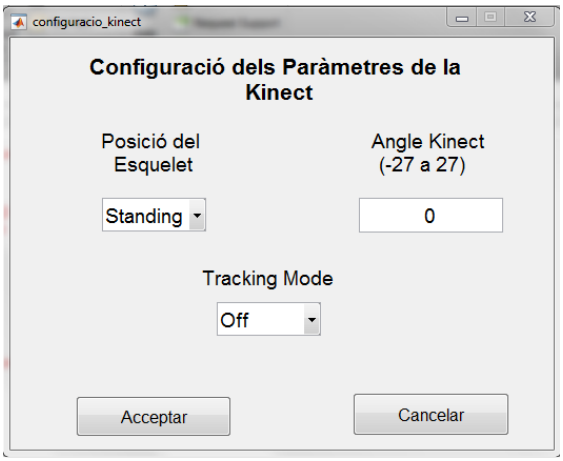

Figura 4.14. Finestra de configuració de la Kinect.

"configuració_kinect" permet configurar si es vol recollir dades de tot l'esquelet; de la part superior, és a dir; braços, espatlles i cap; o detectar una posició concreta. També permet configurar l'angle de la Kinect i el mode per agafar dades "Tracking Mode". En aquest programa tan sols s'utilitzarà el mode "Skeleton".

Panell Drone
Com ja s'ha dit aquest mostra informació de les dades de navegació del drone i a més, tenim un botó anomenat "Connectar Drone" per connectar-nos o desconnectar-nos.

4.3.3 Funcions principals

El programa consta d'una part principal que dóna funcionalitat a la interfície del programa. Així com el que ha de fer quan es polsa cada un dels polsadors o quan es tanca el mateix programa. A més d'aquestes funcions, també s'han creat d'altres per tal de processar les dades que s'han obtingut de la Kinect. És a dir, totes aquelles que calculen els angles dels braços i les dues espatlles.

El codi que es mostrarà a continuació fa referència al descrit al primer paràgraf d'aquest apartat. Com que el codi del programa és molt extens tan sols es mostrarà aquell que s'ha cregut adient per a la comprensió del funcionament del programa.

Una vegada realitzada la configuració adient de la Kinect i haver-li donat al polsador "Connectar". Un cop polséssim el botó Agafar Dades, el programa establirà connexió amb la Kinect i si hem polsat al polsador "Connectar Drone", també ho farà amb aquest. Si no és el cas, tan sols es mostrarà les dades captades per la Kinect si detecta algun esquelet. El codi que conté la funció del polsador "Agafar Dades" és el següent:

```
function buttonAgafarDades_Callback(hObject, eventdata, handles)

global stateButton elAngle erAngle shlAngle shrAngle shlYawAngle...
    shrYawAngle i takenoff vel ang dr
   handles.stateAgafarDades = get(hObject,'String'); % Comproba
l'estat del
   ...boto stop si esta en modo STOP el programa executara el while
   guidata(hObject,handles); % Comproba l'estat del estring agafar
dades
   stateButton = handles.stateAgafarDades;
 if strcmp(handles.stateButtonConnectar,'Desconnectar')
   if strcmp( stateButton ,'Agafar Dades')
```

```
        set(handles.mostrarMissatges,'String','');
        set(hObject,'String','STOP');
          dr = ARDrone;
        start(handles.depth);
   else
        set(hObject,'String','Agafar Dades');
        delete dr;
   end
 else
   set(handles.mostrarMissatges,'String','Error: Clica al boto
Connectar');
 end

   handles.stateAgafarDades = get(hObject,'String'); % Comproba
l'estat
   ...del boto stop si esta en modo STOP el programa executara el
while
   guidata(hObject,handles); % Actualitzem la GUI
   stateButton =  handles.stateAgafarDades;
   i = 1;
   takenoff = 0;
   vel = 0.5; % Velcoitat del drone sobre el pla horitzontal
   ang = 1; % velocitat de rotació i moviment sobre el pla vertical
     %--Inicialisation
    timeArray(1) = 0;
   frameRate=tic;
   displayTime = tic;
   timeDisplayNavData = tic;
```

El codi anterior, s'encarrega d'inicialitzar les variables que s'utilitzaran en aquesta funció i a més fer les comprovacions per tal d'assegurar-nos que s'ha configurat la Kinect i ens hem connectat a aquesta. El següent codi mostra el contingut del bucle principal. Aquest serà dividit en tres parts per tal que el lector comprengui una mica millor el contingut:

```
while  strcmp(stateButton,'STOP')
     trigger(handles.depth);
    [~,timeData,metaData]=getdata(handles.depth); % Adquirim les dades
    ...de la kinect

        timeArray(2)=timeData;
        time = (timeArray(2)-timeArray(1));
        FrameRate = round(1/time);
        timeArray(1)=timeArray(2);

        if toc(frameRate)>=0.2 %Mostrem el Frame Rate cada 0.2 s
             %Kinect Frame Rate
             set(handles.frameRate,'String',...
                 [num2str(round(handles.depthSrc.FrameRate)) '/30
fps']);
             %Matlab Frame Rate
             set(handles.textFrameRateMtb,'String',...
                 [num2str(round(FrameRate)) ' fps']);
            frameRate=tic;
        end
```

El codi anterior s'encarrega de captar les dades del sensor de profunditat i mostrar-lo per pantalla. Aquest també mostra els fotogrames per segon tant de la Kinect com del Matlab amb un període de refresc de 0,2 s.

```
if sum(metaData.IsSkeletonTracked) > 0 % Comproba si detecta
l'esquelet
        realCoordSkeleton= metaData.JointWorldCoordinates(:,:,...
            metaData.IsSkeletonTracked); % Llegim les dades en...
        ...coordenades reals
       %--Angle colze esquerre
            elAngle(i)=angleColzeEsquerre(realCoordSkeleton);
            elEnter = int16(elAngle);
            eL = elEnter(i);
       %--Angle colze dret
            erAngle(i) = angleColzeDret(realCoordSkeleton);
            erEnter = int16(erAngle);
            eR = erEnter(i);
      %--Angle balanceig hombro esquerre
            shlAngle(i) = angleHombroEsquerre(realCoordSkeleton);
            shlEnter = int16(shlAngle);
            shl = shlEnter(i);
      %--Angle balanceig hombro dret
            shrAngle(i) = angleHombroDret(realCoordSkeleton);
            shrEnter = int16(shrAngle);
            shr = shrEnter(i);
      %--Angle guinyada hombro esquerre
            shlYawAngle(i) = angleYawHombroEsquerre(realCoordSkeleton);
            shlYawEnter = int16(shlYawAngle);
            shlY = shlYawEnter(i);
      %--Angle guinyada hombro esquerre
            shrYawAngle(i) = angleYawHombroDret(realCoordSkeleton);
            shrYawEnter = int16(shrYawAngle);
            shrY = shrYawEnter(i);
                if handles.takeoff == 0 &&...

strcmp(get(handles.buttonConnectarDrone,'String'),...
                        'Desconnectar')% Si no ha despegat que ho
fagi
                    dr.takeoff(); % Fer despegar el drone
                     handles.takeoff = 1;
                end
                if handles.takeoff == 1
                  if (shl>=80 && shl<=110 && shr<20) % Comprovem si
                  ...el angle del hombro esquerre amb la vertical fa
                  ...aproximadament 90 graus i el hombro dret està en
repòs
                     dr.rotateLeft(ang); %Fem rotar el dron a
l'esquerra
                  end
                  if (shr>=80 && shr<=110 && shl<20) %Comprovem si
                  ...el angle del hombro dret amb la vertical fa
                  ...aproximadament 90 graus i el hombro esquerre està
                  ...en repòs
                      dr.rotateRight(ang); %Fem rotar el dron a la
dreta
                  end
                  if (shl>=130 && shr>=130) % Si els braços estan
aixecats
                      ...cap d'alt el drone Puja
                     dr.moveUp(ang);
                  end
                  if ( shl >= 80 && shl <= 110 )&&...
                          (shr>=80 && shr<=110)&&(shlY<=30 && shrY<=30)
                      % Si els braços estan a 90 graus el drone baixa
```

```
                    dr.moveDown(ang);
                end
                if (shl>=65 && shl<=110) && (shr >= 65 && shr <= 110
)...
                        && ( shlY >= 80 && shrY >= 80) % Si els
braços
                    ...estan estirats cap endavant
                   dr.moveForward(vel); % El drone es mou cap
endavant
                end
                if (shl<=50)&&(shr<=50)&&(shlY>=80 && shrY>=80)&&...
                        (eL<=50 &&  eR<=50)% Si els braços estan
arronsats
                    ...el drone es mou cap endarrere
                   dr.moveReverse(vel);
                end
                if (shl>=70 && shl<=110) && (shr>=70 && shr<=110)...
                        && (eL<=100) && (eR>=120)%Si el braç dret
està
                    ...estirat i el colze esquerre forma un angle més
                    ...petit de 100° movem cap a la dreta
                   dr.moveRight(vel);
                end
                if (shl>=70 && shl<=110) && (shr>=70 && shr<=110)...
                        && (eR<=100) && (eL>=120)
                   dr.moveLeft(vel);
                end
             end
          if  toc(displayTime)>=0.5
                   %--Angle colze esquerre
                       set(handles.elbowLeft,'String',num2str(eL));
                   %--Angle colze dret
                       set(handles.elbowRight,'String',num2str(eR));
                  %--Angle guinyada hombro esquerre

set(handles.shoulderLeft,'String',num2str(shl));
                  %--Angle guinyada hombro dret

set(handles.shoulderRight,'String',num2str(shr));
                  %--Angle capcineig hombro esquerre
                       set(handles.yawLeft,'String',num2str(shlY));
                  %--Angle capcineig hombro esquerre
                       set(handles.yawRight,'String',num2str(shrY));
             displayTime=tic;
          end
    elseif handles.takeoff == 1
        dr.land(); % Aterrar el Drone
        handles.takeoff = 0;
    end
```

El codi anterior s'encarrega de processar la informació obtinguda pel sensor de profunditat de la Kinect i calcular els angles. Un cop calculats i mitjançant una sèrie de condicionals s'envia la informació al drone. En cas que es compleixi algun d'aquest, el drone farà el moviment adient. Per exemple, si estirem els dos braços cap endavant, se li enviarà la comanda perquè aquest vagi cap endavant. Aquesta part del codi també s'encarrega de mostrar per pantalla cada un dels angles amb un període de refresc de 0.5 s.

Com es pot observar, el càlcul de cada un dels angles té una funció dedicada. Com el codi és molt similar a tots els casos, n'explicarem tan sols un. El codi que es mostrarà a continuació, calcula l'angle de l'espatlla esquerra.

```
function [ shlAngle ] = angleHombroEsquerre( realCoordSkeleton )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
       shlPoint = realCoordSkeleton(5,:,1); % Ens dona el valor del
punt
       ...del espatlla
       elPoint = realCoordSkeleton(6,:,1); % Ens dona el valor del
colze esquerre
       constantVector = [0 1 0];
       elShlVector = shlPoint - elPoint; % Calcula les components del
       ...vector colze i espatlla esquerre
        %producte escalar dels dos vectors que surten del colze
       producteEscalar_elAngle=dot(elShlVector, constantVector);
       %Calcula l'angle en radians del colze mitjançant la formula
       %del producte escalar
       angleRadians = acos(producteEscalar_elAngle/...
           (norm(elShlVector,2)*norm( constantVector,2)));
       shlAngle = angleRadians*180/pi;
end
```

El procediment que s'ha utilitzat per a calcular els angles és el mateix que s'hi ha empleat per a calcular els angles amb el Simulink. Però en aquest cas, implementat en codi Matlab. Podem observar que proporcionem a la funció un array "realCoodSkeleton" de tres dimensions, el qual conté les dades de cada una de les parts de l'esquelet detectat. En aquest cas, agafem les dades de l'espatlla i colze esquerre i calculem el vector entre aquests dos punts. A més, necessitarem un vector que passi per l'eix Y. Aquest es defineix de la següent manera `constantVector = [0 1 0]`.

Ara que ja disposem dels dos vectors, es realitza el producte escalar entre aquests dos i finalment, obtenim l'angle guinyada de l'espatlla esquerra en radians. Per tant, tan sols cal passar de radians a graus i obtenim l'angle desitjat.

4.3.4 Obtenció de les dades de navegació del drone

Aquesta part del codi s'encarrega d'agafar les dades de navegació del drone i mostrar-les per pantalla. Abans de continuar, es recomana consultar l'apartat dels annexes: "Classe per a connectar-se al AR Drone amb Matlab".

```
if strcmp(get(handles.buttonConnectarDrone,'String'),'Desconnectar')
                  if toc (timeDisplayNavData) >=0.5
                     set(handles.textBateria,'String',...
                        [num2str(dr.Battery_Voltage) '%']);
                     set(handles.textPitch,'String',...
                        [num2str(round(dr.Pitch)) '°']);
                     set(handles.textRoll,'String',...
                        [num2str(round(dr.Roll)) '°']);
                     set(handles.textYaw,'String',...
                        [num2str(round(dr.Yaw)) '°']);
                     set(handles.textAltitud,'String',...
                        [num2str(round(dr.Altitude,3)) ' m']);
                     set(handles.textVx,'String',...
                        [num2str(round(dr.X_Velocity,4)) ' m/s']);
                     set(handles.textVy,'String',...
```

```
                    [num2str(round(dr.Y_Velocity,4)) ' m/s']);
                timeDisplayNavData=tic;
            end
    end
```

El temps de refresc per actualitzar les dades mostrades per pantalla és de 0,5 s. A continuació anomenarem les dades que es mostren per pantalla:

- Estat de la bateria.
- Angles balanceig, capcineig i guinyada.
- Altura del Drone.
- Velocitats en l'eix de coordenades x i y.

4.3.5 Resultats obtinguts

Per tal d'il·lustrar el funcionament del programa s'ha decidit guardar les dades, ja que d'aquesta manera podrem calcular la trajectòria que ha seguit el drone. A més a més, es té una idea de l'eficaç que és el control. Cal remarcar però que aquest tan sols ens proporciona les dades de l'altura i les velocitats en els eixos x i y. Per tant, s'ha calculat el temps transcorregut entre mostra i mostra i s'ha calculat la posició punt per punt mitjançant les següents fórmules:

- **Coordenades de l'eix x**

$$x_i = x_{i-1} + v_{x(i-1)} * \Delta t + \frac{1}{2}\Delta v_x * \Delta t \qquad (2)$$

On:

- x_i*: Coordenada actual sobre l'eix x.*
- x_{i-1}*: Coordenada anterior sobre l'eix x.*
- Δv_x*: Diferència de velocitats entre el punt actual i l'anterior de l'eix x.*

- **Coordenades de l'eix y**

$$y_i = y_{i-1} + v_{y(i-1)} * \Delta t + \frac{1}{2}\Delta v_y * \Delta t \qquad (3)$$

On:

- y_i*: Coordenada actual sobre l'eix y.*
- y_{i-1}*: Coordenada anterior sobre l'eix y.*
- Δv_y*: Diferència de velocitats entre el punt actual i l'anterior de l'eix y.*

La figura 4.15 mostra la trajectòria que ha seguit el drone. El punt verd mostra on ha s'ha enlairat i el vermell on ha aterrat.

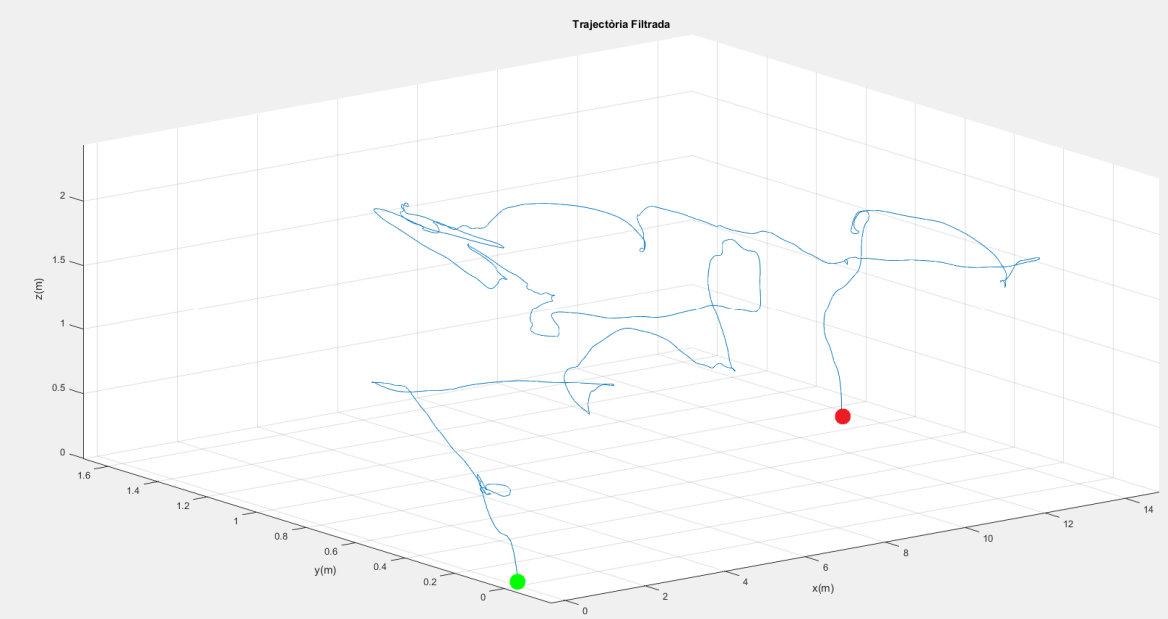

Figura 4.15. Dades de la trajectòria del Drone filtrades.

Es pot observar a la figura anterior que les dades han estat filtrades mitjançant un filtre passa baix Butterworth[7] a una freqüència de tall de 5 Hz, ja que la freqüència de mostreig del drone és de 30 Hz. En aquest cas tan sols es pretén que el lector tingui una idea de la trajectòria que ha seguit sense tenir en compte la distància real que ha recorregut. La següent figura mostra les dades sense filtrar:

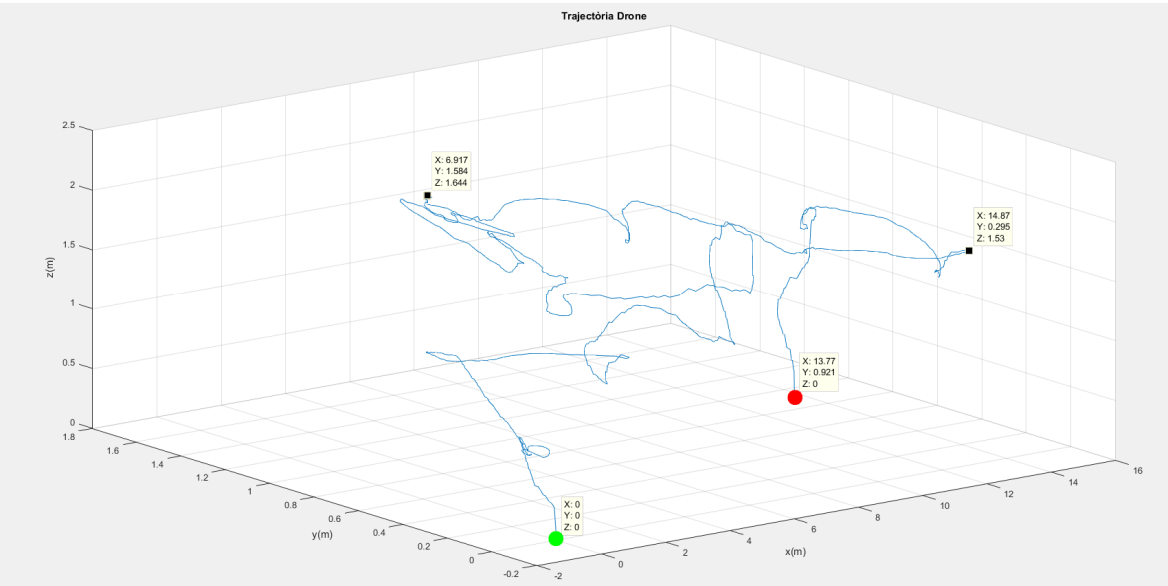

Figura 4.16. Dades de la Trajectòria del Drone.

En aquest cas, hem assenyalat els màxims absoluts que ha recorregut el drone en els tres eixos de coordenades. L'objectiu de les figures 4.15 i 4.16 era fer anar l'aparell en totes direccions. D'aquesta manera es pot observar el control de la Kinect.

[7] Filtre Butterworth: és un tipus de filtre que intenta obtenir la resposta més plana possible fins a la freqüència de tall.

4.3.6 Instruccions per a l'usuari

L'objectiu d'aquest apartat és proporcionar una petita guia per fer funcionar el programa. Aquest és prou intuïtiu, però de totes maneres es recomana consultar aquest punt.

Inicialment, fem córrer el programa i ens l'obrirà amb la interfície gràfica principal. Un cop tenim oberta aquesta finestra, el primer que hem de fer és configurar la Kinect, en cas que es vulgui captar dades. Si tan sols volem mostrar la trajectòria, no és necessari fer res del que es descriurà a continuació. Per tant, cliquem al botó "Configuració". Vegeu la següent figura:

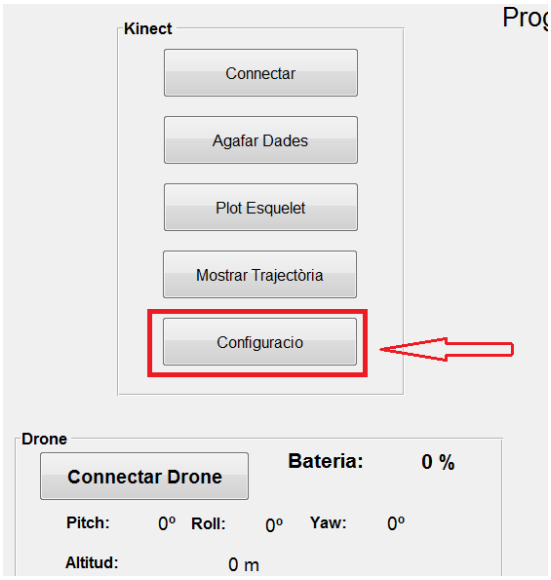

Figura 4.17. Captura de pantalla del panell de botons del programa de la Kinect.

Ens apareixerà la finestra de configuració i configurem els diferents paràmetres com mostra la figura que següent:

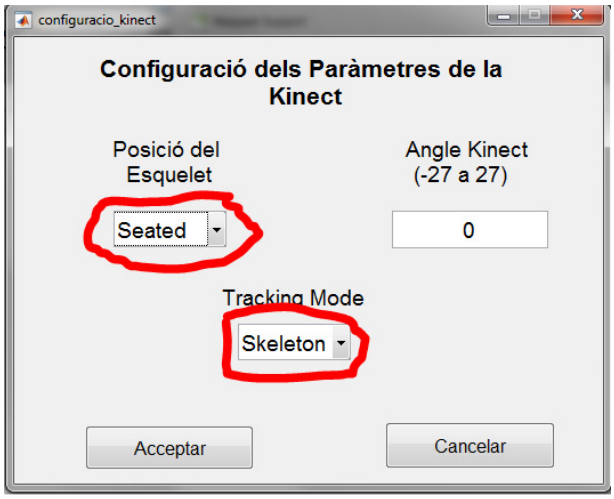

Figura 4.18. Captura de pantalla que mostra la configuració de la Kinect.

Com es pot veure, configurarem la posició de l'esquelet "Seated" i el "Tracking Mode" seleccionem "Skeleton". Cal dir que l'angle s'ha d'ajustar al valor que es cregui adient. Per tant, tan sols cal donar al botó "Acceptar".

Finalment, tan sols cal donar al botó "Connectar Drone", si es vol controlar-lo, polsar al botó "Connectar" i per últim, tan sols cal clicar al botó "Agafar Dades". Una vegada clicat aquest, ja ens podem situar a davant de la càmera i controlar el drone.

Un petit a punt a destacar pel que fa al botó "Connectar Drone" és el següent: encara que el cliquéssim, Matlab no es connectarà amb el drone fins que no es connecti amb la Kinect i es cliqui al botó "Agafar Dades".

5 Leap Motion

En aquest apartat es pretén explicar totes les proves realitzades amb el Leap Motion per tal d'arribar a realitzar el control tant del drone, com desenvolupar aplicacions senzilles per Arduino. Cal destacar que les dades que captem per controlar el drone no són de la mà sinó del dit del mig. Això és a causa del programa que ens proporciona Perry. J.[2], ja que no calcula la posició d'aquesta. De totes maneres els resultats han estat favorables i s'ha aconseguit els dos objectius.

Es poden consultar els resultats del control del drone i l'Arduino mitjançant el dispositiu Leap Motion a l'apartat annexes: "Enllaços dels vídeos del Leap Motion".

5.1 Proves realitzades

Per tal d'elaborar el programa final, s'ha hagut de realitzar una sèrie de proves per tenir una idea dels límits del dispositiu. D'aquesta manera es traçarà un rang d'operació eficaç i segur, per tal de realitzar tant el control del drone, com aplicacions per Arduino. Per captar aquestes dades s'ha realitzat un "Script" que s'encarrega de crear dos gràfics que mostren la posició del dit del mig.

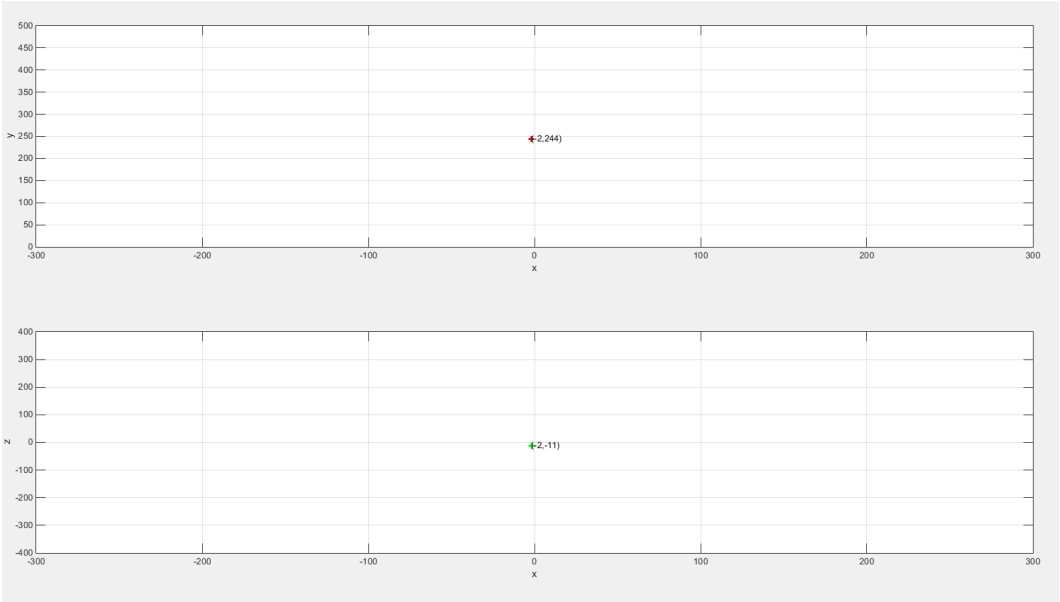

Figura 5.1. Captura de pantalla del "Script" que mostra la posició del dit del mig d'una mà.

La zona segura s'ha traçat a una altura d'entre 60 i 80 mm sobre l'eix y. Això permet que els valors puguin variar entre -250 i 250 mm sobre l'eix x i entre -220 i 220 mm sobre l'eix z.

5.2 Programa final per al control de l'AR Drone

Finalitzat l'anàlisi del Leap Motion i concloure que és factible realitzar el control del Drone mitjançant aquest dispositiu, s'ha procedit a crear el que serà el programa final. Per tant, aquest apartat estirà dedicat a descriure el resultat final del programa, així com la interfície gràfica, els moviments i el codi que conté per realitzar totes les funcions bàsiques de control.

5.2.1 Explicació del control

Els moviments que s'han escollit per realitzar el control, no han estat escollits a l'atzar sinó que han estat analitzats i s'han escollit a consciència. En primer lloc, perquè el drone s'enlairi, s'ha de mostrar al Leap Motion dos dits durant un instant de temps. Si volem que aterri, es fa el mateix. En segon lloc, per fer moure el drone cap endavant, cap endarrere, cap a la dreta o cap a l'esquerra; tan sols cal situar la mà sobre el Leap Motion i desplaçar-la. Sempre conservant la direcció i sentit dels dits cap endavant del dispositiu. És a dir, si desplacem la mà cap endavant respecte al centre del dispositiu, el drone es mourà endavant i el mateix per a la resta de moviments. En cas que la punta dels dits estigui situat al mig del dispositiu, el drone romandrà a la mateixa posició.

Per una altra banda, per fer pujar o baixar el drone es té en compte si la mà i els dits estan mirant amunt o avall, respectivament. Per fer-lo rotar sobre si mateix, tan sols cal rotar la mà cap al sentit que volem que aquest giri. Per tant, si volem per exemple que el drone giri cap a la dreta, girem la mà cap a la dreta.

Finalment, cal remarcar que si el drone s'ha enlairat i està en moviment, en cas que el dispositiu Leap Motion deixi de detectar la mà, el drone deixarà de realitzar el moviment que estava fent i quedarà suspès a l'aire. Inicialment, el programa es va dissenyar que si el Leap Motion detectava una mà el drone s'enlairava i si deixava de detectar-la, aquest aterrava. Després de realitzar una sèrie de proves, em vaig adonar que per aquest dispositiu no era convenient fer-ho d'aquesta manera descrita. El motiu bàsic d'aquest canvi va ser que sempre estenia un impuls a treure la mà quan el drone anava a col·lidir contra una paret o objecte. Per més informació, es recomana consultar l'apartat annexes: "Glossari de gestos amb el Leap Motion per al control de l'AR Drone".

5.2.2 Interfície Gràfica

De la mateixa manera que el programa creat per controlar el drone amb la Kinect, s'ha tingut en compte que el temps de resposta sigui el menor possible entre el Leap Motion i el drone. Per tant, s'ha evitat sobre carregar la interfície gràfica. Però sense perdre eficàcia alhora de mostrar les dades que capta el Leap Motion i les dades que rep de l'AR Drone. Observem la figura 5.2:

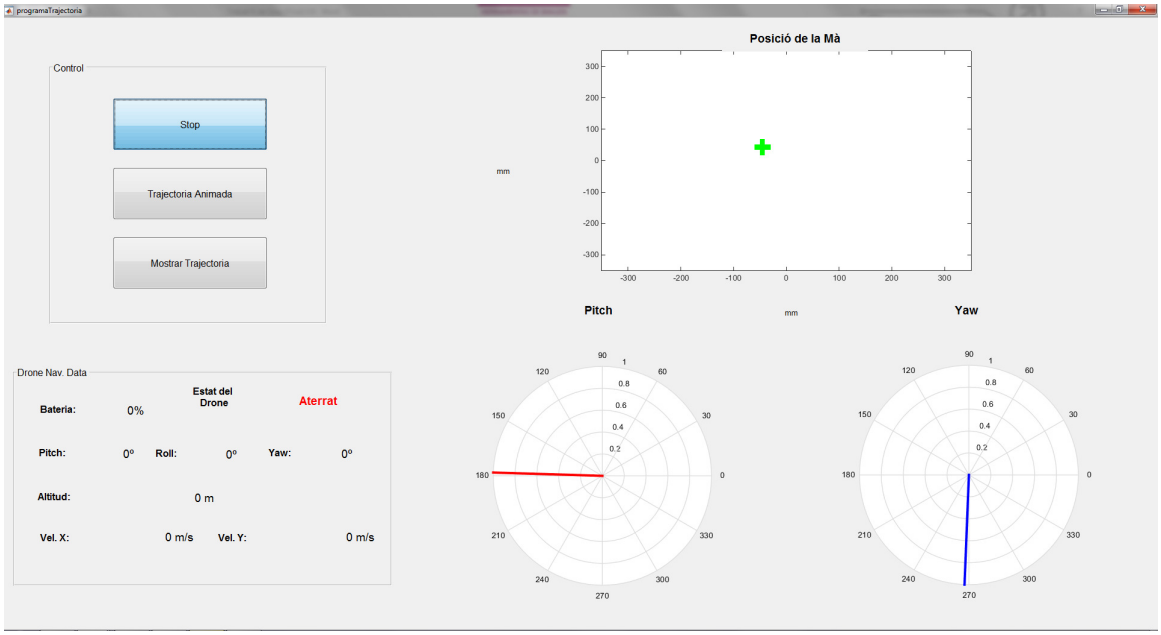

Figura 5.2. Captura de pantalla de la interfície del programa Leap Motion.

Podem observar que el programa disposa d'un panell de control on hi ha tres botons:

- **Start/Stop:** aquest s'encarrega de connectar-se tant amb el Leap Motion com amb el drone, obtenir dades dels dos dispositius i per descomptat el control de l'AR drone.

- **Trajectòria Animada:** cada vegada que haguéssim fet volar el Drone, les dades de navegació seran mostrades mitjançant un gràfic en tres dimensions on mostra la trajectòria de manera animada.

- **Mostrar Trajectòria:** té el mateix objectiu que el botó anterior però en aquest cas mostra de cop tota la trajectòria. Això s'ha afegit perquè si es realitzen trajectòries llargues, mitjançant el cas anterior triga una bona estona en dibuixar-la tota.

El programa també disposa d'un panell on es mostra l'estat del drone i les dades que aquest envia. Vegeu la figura 5.3:

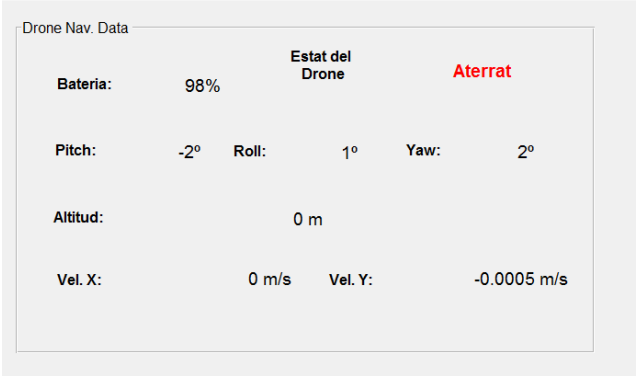

Figura 5.3. Captura de pantalla del panell de dades del drone.

Finalment, el programa disposa de tres gràfics, el primer dels tres mostra la posició del dit del mig i els dos restants s'encarreguen de mostrar mitjançant un vector la direcció d'aquest. D'aquesta manera l'usuari que fa ús del programa, té una idea de la inclinació de la mà, és a dir, l'angle capcineig i a més a més, de l'angle guinyada.

5.2.3 Codi Matlab

En aquest apartat es descriuran les funcions principals que s'encarreguen de processar la informació del Leap Motion i controlar el drone. L'estil per crear el codi ha estat similar al que s'ha realitzat al programa per a la Kinect. En general, podem resumir que el programa disposa d'un bucle principal, encarregat d'agafar les dades del sensor, processar-les i enviar-les al drone. El bucle també conté el codi per a recollir les dades de navegació i mostrar-les per pantalla. Per tant, entrem una mica més en detall:

Una vegada cliquem al botó "Start", el programa inicialitza les variables que s'utilitzaran. Fet això, entrem al bucle principal. Primer de tot s'adquireix un fotograma de dades del Leap Motion mitjançant la següent assignació:

```
metaData = matleap(1)
```

En cas que "metaData.pointables" contingui dades, el programa realitza un sumatori dels dits allargats i actualitza les variables x i z; i les components u i v.

```
for i=1:5 %Bucle per sumar quants dits hi ha allargats
          sumaDits = sumaDits+metaData.pointables(i).is_extended;
end
```

```
        x(j)=metaData.pointables(3).position(1);
        z(j)=metaData.pointables(3).position(3);
        x(j) = round(x(j));
        z(j) = round(z(j));

        % Control Yaw and Pitch del dit del mig
        u(j)=metaData.pointables(3).direction(1);
        v(j)=metaData.pointables(3).direction(2);
        w(j)=metaData.pointables(3).direction(3);
        u(j) = round(u(j),2);
        v(j) = round(v(j),2);
        w(j) = round(w(j),2);
```

A continuació, comprova si hi ha dos dits allargats. En cas que sigui així i el drone està aterrat, s'enlaira i a l'inrevés. Per realitzar el control del drone, el programa comprova si hi ha allargats més de dos dits. Si és així, procedeix a realitzar els següents càlculs:

```
%---Realitzar control del drone--------------------
     if despegat == 1 && sumaDits > 2 %Si tots els dits extesos
      %Anar cap a la dreta
      if x(j) > 70
          rightTilt = 0.0146*x(j)-0.9231; % 0.1<=rightTilt<=2
          ...and 70<x<=200
          if rightTilt > 2
              rightTilt = 2;
          end
          ar.moveRight(rightTilt);
          is_Moving = 1;
      %Anar cap a l'esquerra
      elseif x(j) < -70
          leftTilt = -0.0146*x(j)-0.9231; % -0.1<=leftTilt<=-2
          ...and -70<x<=-200
          if leftTilt < -2
              leftTilt = -2;
          end
          ar.moveLeft(leftTilt);
          is_Moving = 1;
      end
      %Anar cap endavant
      if z(j) > 45
          reverseTilt =  0.0123*z(j)-0.4516; % 0.1<=reverseTilt<=2 and
45<z<=200
          if reverseTilt > 2
              reverseTilt = 2;
          end
          ar.moveReverse(reverseTilt);
          is_Moving = 1;
      %Anar cap endarrere
      elseif z(j) < -70
          forwardTilt =  -0.0146*z(j)-0.9231; % -0.1<=reverseTilt<=-2
          ...and -70<z<=-200
           if forwardTilt < -2
              forwardTilt = -2;
          end
          ar.moveForward(forwardTilt);
          is_Moving = 1;
      end
```

Com es pot observar, el control de l'angle "pitch" i "yaw" és variable en funció de la posició de la mà, és a dir, la velocitat dels moviments del drone varia en funció de la posició i direcció d'aquesta. Per exemple, si desplacem la mà sobre l'eix z, com més creixi el valor de z més velocitat adquirirà el drone, ja que l'angle "pitch" augmenta. Per tant, ens permet tindre un millor control del drone. Observem que deixem uns marges en els valors de x i z mitjançant condicionals. El motiu d'això és per deixar un marge de seguretat per tal que no es consideri moviment si no se superen aquests límits. El mateix es pot dir que hem fet per les components u i v.

$$-70 \leq x \leq 70$$

$$-70 \leq z \leq 45$$

$$-0{,}4 \leq u \leq 0{,}4$$

$$-0{,}4 \leq v \leq 0{,}4$$

En resum, si la mà romana dintre dels valors anomenats anteriorment, el drone no realitzarà cap moviment.

El codi següent és molt semblant a l'anterior però en aquest cas s'encarrega de calcular els diferents valors per a fer rotar el drone i per fer pujar-lo o baixar-lo:

```
%Rotar cap a la dreta
if u(j) > 0.4
    yaw = 3.1667*u(j)-1.1667; % 0.2<=yaw<=2 and  0.4<u<1
    ar.rotateRight(yaw);
    is_Moving = 1;
%Rotar cap a l'esquerra
elseif u(j) < -0.4
    yaw = -3.1667*u(j)-1.1667;  % 0.2<=yaw<=2 and  -0.4<u<-1
    ar.rotateLeft(yaw);
    is_Moving = 1;
end
%Anar amunt
if v(j) > 0.4
    pitch = 3.1667*v(j)-1.1667; % 0.2<=pitch<=2 and  0.4<v<1
    ar.moveUp(pitch);
%Anar avall
elseif v(j) < -0.4
    pitch = -3.1667*v(j)-1.1667; %0.2<=pitch<=2 and  -0.4<v<-1
    ar.moveDown(pitch);
    is_Moving = 1;
end
end
```

Observem que els angles que obtenim en els càlculs per a enviar-li, són proporcionals a les dades obtingudes pel sensor Leap Motion. Per realitzar aquesta relació lineal entre les dades obtingudes pel sensor i els valors que enviem al drone, s'ha tingut en compte uns límits mínim i màxim d'operació. Per tal de comprendre millor el descrit, s'exposarà un exemple. Observem la part del codi on es processa les dades de l'eix x:

```
rightTilt = 0.0146*x(j)-0.9231; % 0.1<=rightTilt<=2 and
70<x<=200
if rightTilt > 2
```

```
        rightTilt = 2;
    end
    ar.moveRight(rightTilt);
```

En concret observem l'equació de la recta per determinar l'angle balanceig per fer anar el drone a la dreta:

```
rightTilt = 0.0146*x(j)-0.9231;
```

Aquesta equació s'ha determinat per uns valors de x que varien entre 70 i 200 mm. Per tal que el valor de "rightTilt" variï entre 0,1 i 2. Cal destacar que els valors de l'angle balanceig que se li envien, han d'estar entre -1 i 1 però la funció de l'objecte "`ar.moveRight(rightTilt)`" ja realitza la conversió. La figura 5.4 ens mostra el segment de l'equació de la recta en una gràfica.

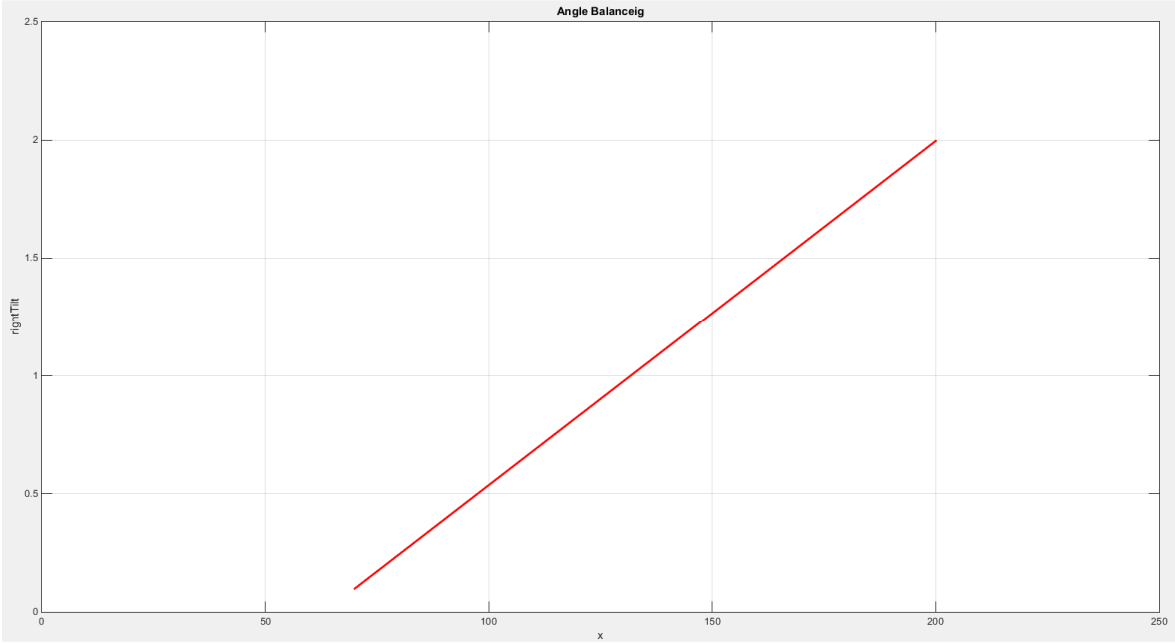

Figura 5.4. Gràfica del segment per determinar el valor de "RightTilt".

Finalment, els botons "Trajectòria animada" i "Mostrar Trajectòria" criden a dos "Scripts" per dibuixar la trajectòria i mostrar-la sencera, respectivament. Per a més informació relacionada amb el codi del programa es recomana consular l'apartat annexes: "Programa Principal".

5.2.4 Obtenció de les dades de navegació del Drone

El procediment per obtenir les dades de navegació ha estat exactament el mateix que en el cas de la Kinect, per aquest motiu no s'entrarà en més detall. Per a més informació es recomana consultar el codi de la classe "ARDrone" a l'apartat annexes: "Classe per a connectar-se al AR Drone amb Matlab".

5.2.5 Resultats obtinguts

Com en el cas de la Kinect, per al Leap Motion també s'ha realitzat proves i s'han registrat les dades per mostrar la trajectòria d'aquest. El procediment que s'ha utilitzat per al càlcul d'aquesta és idèntic que en el cas de la Kinect per tant no es tornarà a explicar.

Per tal d'il·lustrar al lector amb un exemple clar s'ha procurat fer una trajectòria simple. En aquest cas s'ha realitzat una trajectòria que defineix la forma d'un quart de cercle. A més, les dades inclouen l'enlairament i l'aterratge. Com que la prova es va realitzar a l'aire lliure, les dades es veuen afectades pel mateix vent. Per tant, també es mostraran filtrades.

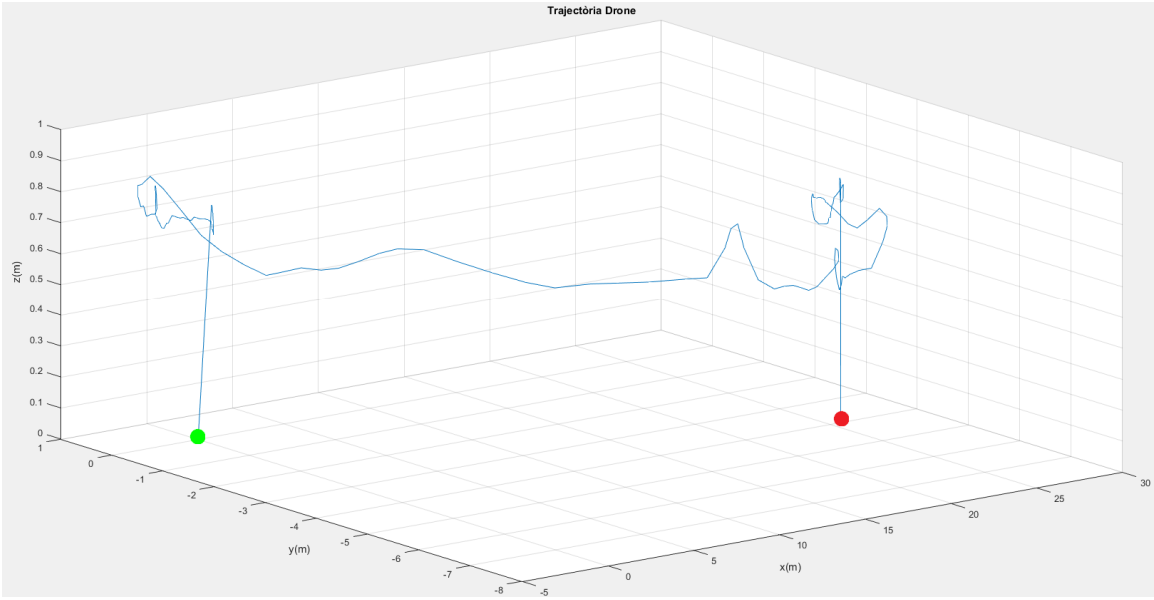

Figura 5.5. Dades de navegació del drone. El punt verd és l'enlairament i el punt vermell l'aterratge.

Es pot observar que els resultats han estat prou satisfactoris. De totes maneres a continuació es mostraran els resultats filtrats per observar millor la trajectòria que aquest ha seguit.

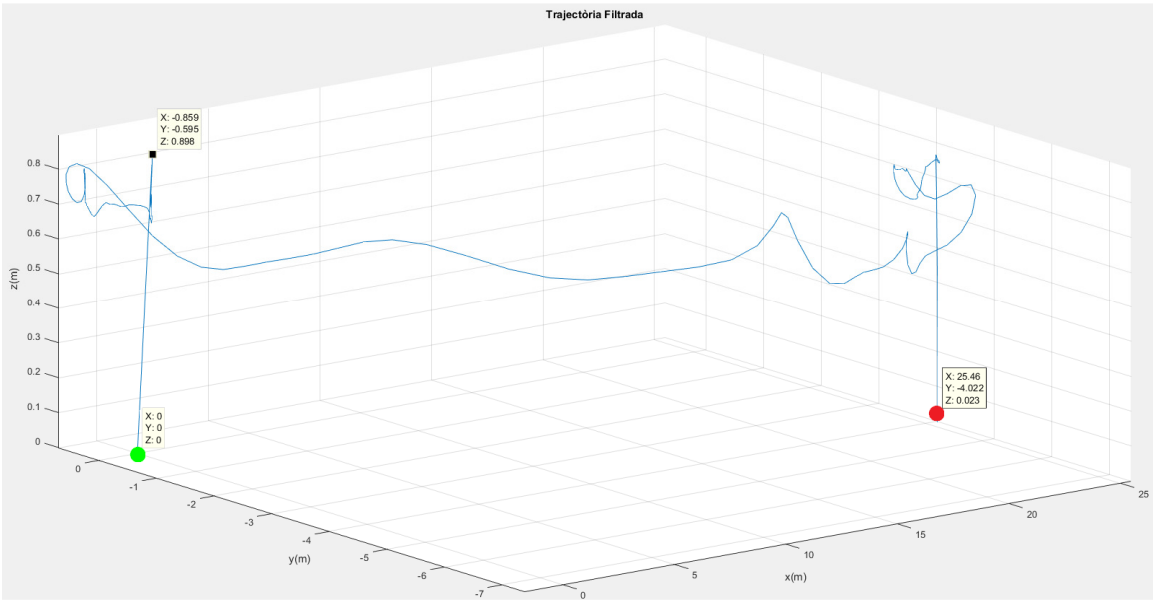

Figura 5.6. Dades de la trajectòria del drone filtrades. El punt verd representa l'enlairament i el vermell l'aterratge.

La freqüència de mostreig del drone és de 30 Hz, per tant per mostrar les dades de la figura prèviament, han estat filtrades mitjançant un filtre ButterWorth a una freqüència de tall de 5 Hz. Un altre detall a destacar és l'altura a la qual va volar. Aproximadament va ser entre 0,6 i 0,9 m.

Finalment, per mostrar l'eficiència de la variació de la velocitat del drone segons la posició de la mà. Es compararà les dades que el programa li envia i el que aquest assoleix. Cal recordar que nosaltres li enviem l'angle d'inclinació[8] que varia entre -1 i 1 i en canvi el drone ens envia dades de la velocitat lineal en els eixos Vx i Vy. Per tant, tan sols es comparà la forma de les gràfiques obtingudes, tant de les dades que enviem com de les que aquest ens retorna. Les dades que s'han recollit per mostrar el descrit anteriorment, s'ha realitzat mitjançant una prova on s'ha intentat que el drone anés el més recte possible sobre l'eix x.

A continuació mostrarem la trajectòria que ha seguit aquest:

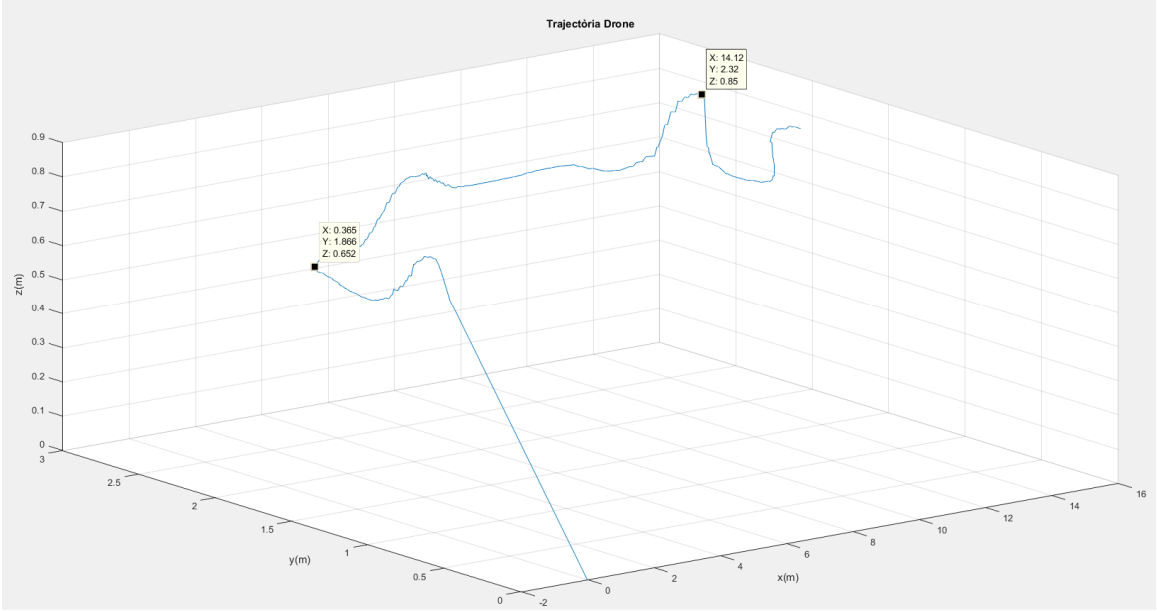

Figura 5.7. Gràfica de la trajectòria per mostrar la velocitat en l'eix x.

A les figures següents es pot observar la velocitat que aquest adquirit durant la trajectòria i l'angle que se li ha enviat:

[8] Angle inclinació o angle pitch: cal dir que el drone sols accepta valors de -1 a 1 però l'arxiu extret d'internet per a la connexió drone-matlab, accepta valors d'entre -2 i 2. Aquest s'encarrega de fer la conversió abans d'enviar-ho al drone.

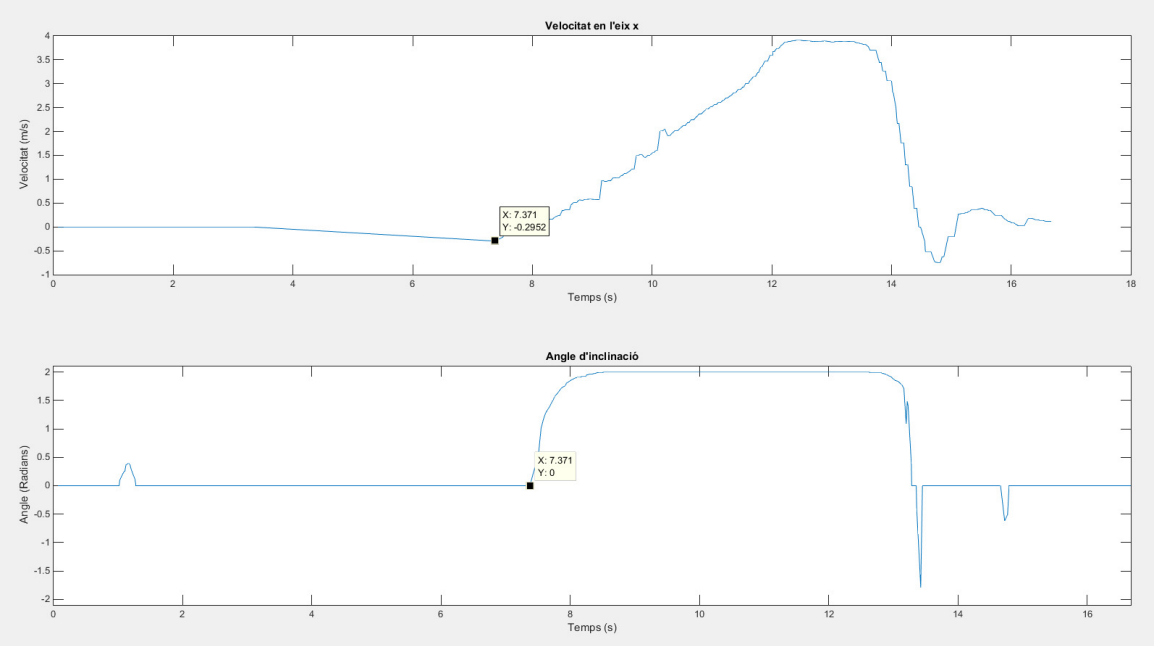

Figura 5.8. Gràfiques de la velocitat en l'eix x i l'angle d'inclinació que se li envia al drone.

En primer lloc, es pot observar que la resposta del drone és quasi immediata en rebre una comanda perquè aquest vagi cap endavant. A més, podem observar que la variació de velocitat funciona correctament. Cal dir però que els límits que hem utilitzat perquè el Leap Motion detecti la mà ens suposen un problema amb el control, ja que tenim molt poc marge de maniobrabilitat. Per tant, movent una mica la mà obtenim la màxima velocitat. Tot i així cal dir que els resultats són prou favorables.

5.3 Aplicacions amb Arduino

Com ja havíem dit, per mostrar l'abast del Leap Motion en aquest cas, s'ha realitzat un programa que compta quants dits de la mà estan allargats i mostra el resultat en un "array" de 5 Leds. Si es mostren més de 5 dits, el programa finalitza. Aquest és simple però eficaç per il·lustrar un altre exemple d'aplicació que es pot realitzar en un altre hardware com és l'Arduino.

5.3.1 Material i dispositius utilitzats

- Leap Motion.
- Arduino UNO.
- LEDs de diferents colors.
- 5 resistències de 220 Ω.
- 1 placa protoboard.

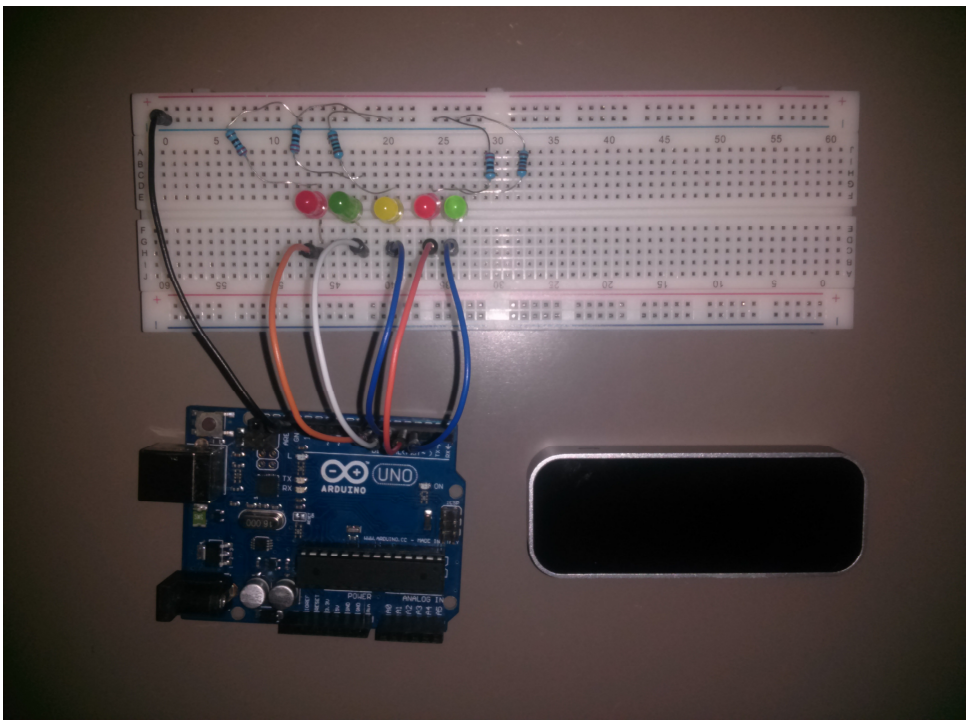

Figura 5.9. Dídac Coll, Imatge del material emprat. 27 de maig de 2015.

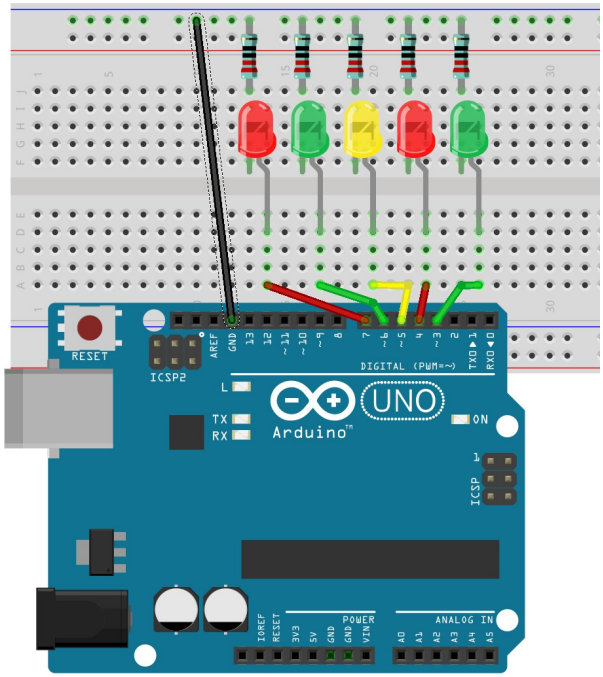

Figura 5.10. Circuit creat amb el programa Fritzing.

La figura 5.10 mostra les diferents connexions dels Leds als pins digitals de l'Arduino UNO. En aquest cas utilitzem del 3r al 7è pin.

5.3.2 Programa

El programa per a controlar l'Arduino mitjançant el Leap Motion també ha estat realitzat en codi Matlab. Matlab, com ja s'havia anomenat, permet interactuar amb el hardware Arduino en temps real. Això ho aconsegueix enviant comandes mitjançant el port sèrie de la placa. D'aquesta manera tota la informació processada que captem amb el sensor l'utilitzem per interactuar amb aquesta. El programa ha estat creat perquè tan sols s'hagi d'utilitzar el sensor una vegada ha iniciat el programa. Ja que com he anomenat anteriorment tan sols cal mostrar més de cinc dits per finalitzar l'execució. A continuació es descriurà els aspectes més importants del codi emprat:

Per tal de començar agafar dades utilitzem un bucle "while", realitzem la següent assignació per agafar les dades del sensor:

```
metaData =  matleap_frame;
```

Una vegada obtingut el fotograma de dades s'analitza si hi ha dades a rastrejar, si és el cas, realitzem el següent bucle "for":

```
for i=1:length(metaData.pointables)
  extendedFingers=extendedFingers+metaData.pointables(i).is_extended;
end
```

Aquest bucle s'encarrega de comptar el nombre de dits. Un cop ha finalitzat, la variable "extendedFingers" conté el valor del nombre de dits allargats. Finalment, un "Switch" és l'encarregat d'escollir l'acció a realitzar en funció del valor de la variable, és a dir, encendre un o més d'un Led, cap o finalitzar el programa. El codi complet es pot trobar a l'apartat d'annexes: "Programa Arduino".

6 Conclusions

Després d'una anàlisi exhaustiu dels resultats i les proves realitzades que s'han dut a terme en aquest projecte de fi de carrera, es pot concloure que s'han arribat als objectius plantejats inicialment.

Per una banda, cal destacar la feina realitzada amb la Kinect i el Leap Motion en aquest treball, ja que hi ha un bon recull d'exemples i guies per fer ús d'aquests dos dispositius amb el programa Matlab. Per tant, qualsevol qui llegeixi aquest treball, no tan sols és capaç de comprendre el funcionament de la Kinect i del Leap Motion sinó que pot realitzar aplicacions diverses mitjançant aquest hardware i Matlab. Si més no, s'ha intentat ser el més curos possible alhora de redactar-lo, per tal que el lector pugui comprendre d'una manera clara i explicita tots els conceptes tècnics que constitueixen aquest treball de fi de grau.

Per una altra banda, treballar amb aquests dos dispositius ha permès comparar-los. En primer lloc, la Kinect ofereix un gran ventall de possibilitats perquè com ja s'ha vist permet calcular dades de tot el cos d'una persona. Aplicant-ho al drone ha permès realitzar un bon control d'aquest. Però tot i que a priori pareix que la Kinect té molts avantatges, cal destacar que té un desavantatge molt important que cal anomenar i és el consum d'energia. Per tant, per realitzar proves amb aquest dispositiu fora del nucli urbà, s'ha de fer ús d'una font d'alimentació externa. En aquest cas s'hauria d'utilitzar una bateria per alimentar el dispositiu per tal que es pogués controlar el drone. En canvi el Leap Motion, tan sols és necessari connectar-lo a un port USB d'un ordenador per fer-lo funcionar. Juntament amb un ordenador portàtil, es converteix en un dispositiu mòbil, el qual permet utilitzar-lo quasi en qualsevol lloc. Cal destacar però, que durant les proves que s'han dut a terme, una d'elles ha estat l'exposició dels dos dispositius als raigs solars. En aquesta prova s'ha comprovat que amb l'exposició directa i fins i tot indirectament en el cas de la Kinect, no funcionen de la manera esperada.

Com ja hem pogut veure, la senzillesa de l'obtenció de dades mitjançant el Leap Motion dels nostres dits i mans permet realitzar aplicacions senzilles amb Arduino. Tot i que es podrien realitzar amb la Kinect s'ha de reconèixer que la comoditat de poder utilitzar un dispositiu assegut des d'una cadira fent ús, tan sols, de les mans és millor que no utilitzar tot el cos.

A més a més, s'ha dut a terme amb èxit totes les proves que s'han realitzat amb el control del drone, tant fent ús de la càmera Kinect com del Leap Motion. Tot i que la interfície gràfica és millorable, s'ha intentat realitzar-la per tal que la comunicació entre el programa i el drone no es vegi afectada per falta de recursos informàtics. A més a més, desenvolupar programari mitjançant la "GUI" de Matlab no tan sols permet a l'autor del treball ampliar els seus coneixements sinó que pot servir de gran ajuda a lectors interessats amb la matèria.

7 Referències

[1] Chikamasa, T. (2011). *Simulink Support for Kinect.*
[2] Perry, J. (s.f.). Extret de Github: https://github.com/jeffsp/matleap
[3] Piskorski, S., Brulez, N., Eline, P., & D'Haeyer, F. (2012). *AR Drone Developer Guide.*

8 Webgrafia

Suport Matlab i Kinect

http://www.mathworks.com/matlabcentral/fileexchange/32318-simulink-support-for-kinect

http://es.mathworks.com/help/imaq/acquiring-image-and-skeletal-data-using-the-kinect.html

Leap Motion

https://www.leapmotion.com/

http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

https://github.com/jeffsp/matleap

AR Drone Parrot

http://ardrone2.parrot.com/

https://projects.ardrone.org/

[4]http://www.mathworks.com/matlabcentral/fileexchange/42532-controller-for-ar-drone/content/ARDrone.m

Arduino

http://www.arduino.cc/

[5]http://es.mathworks.com/matlabcentral/fileexchange/32374-matlab-support-for-arduino--aka-arduinoio-package-

[6]http://www.arduino.cc/en/Main/Software

9 Annexes

Aquest apartat inclou tots els codis que s'han creat, utilitzat i implementat en aquest projecte. A més, es poden trobar els glossaris descriptius dels moviments que s'han creat adients per controlar el drone tant amb la Kinect com amb el Leap Motion.

9.1 Glossari de gestos amb la Kinect per al control de l'AR Drone

Figura 9.1. Dídac Coll. Imatge que mostra la posició del cos perquè el drone es mantingui a la posició que es troba. 27 de maig del 2015.

Figura 9.2. Dídac Coll. Imatge que mostra la posició del cos per fer baixar el drone verticalment. 27 de maig de 2015.

Figura 9.3. Dídac Coll. Imatge que mostra la posició del cos per fer pujar el drone verticalment. 27 de maig del 2015.

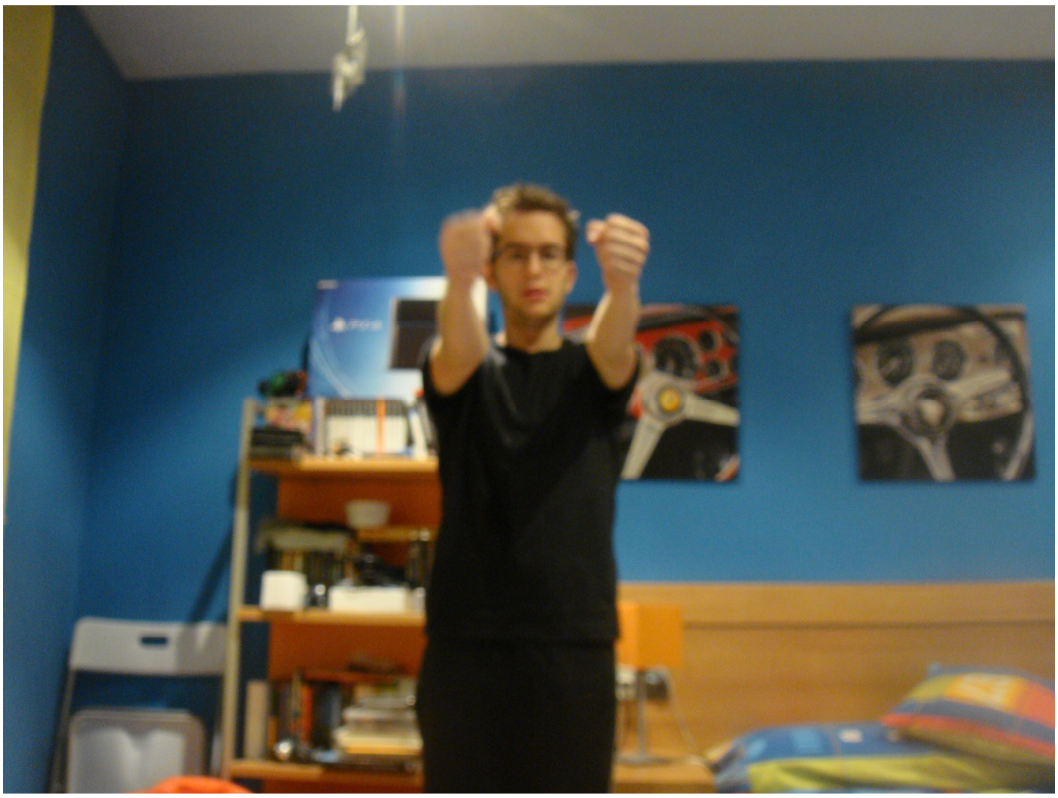

Figura 9.4. Dídac Coll. Imatge que mostra la posició del cos per fer anar endavant el drone. 27 de maig del 2015.

Figura 9.5. Dídac Coll. Imatge que mostra la posició del cos per fer anar el drone endarrere. 27 de maig del 2015.

Figura 9.6. Dídac Coll. Imatge que mostra la posició del cos per fer anar el drone cap a la dreta. 27 de maig del 2015.

Figura 9.7. Dídac Coll. Imatge que mostra la posició del cos per fer anar el drone cap a l'esquerra. 27 de maig del 2015.

Figura 9.8. Dídac Coll. Imatge que mostra la posició del cos per fer rotar el drone sobre si mateix cap a l'esquerra. 27 de maig de 2015.

Figura 9.9. Dídac Coll. Imatge que mostra la posició del cos per fer rotar el drone sobre si mateix cap a l'esquerra. 27 de maig de 2015.

9.2 Glossari de gestos amb per al Leap Motion

El glossari següent consta de diferents fotografies de la mà per mostrar al Lector les diferents posicions de la mà per controlar el drone mitjançant el dispositiu Leap Motion.

Figura 9.10. Dídac Coll. Imatge que mostra la posició de la mà perquè el drone es mantingui a la posició que es troba, 27 de maig del 2015.

Figura 9.11. Dídac Coll, Imatge que mostra la posició de la mà per fer anar endavant el drone, 27 de maig del 2015.

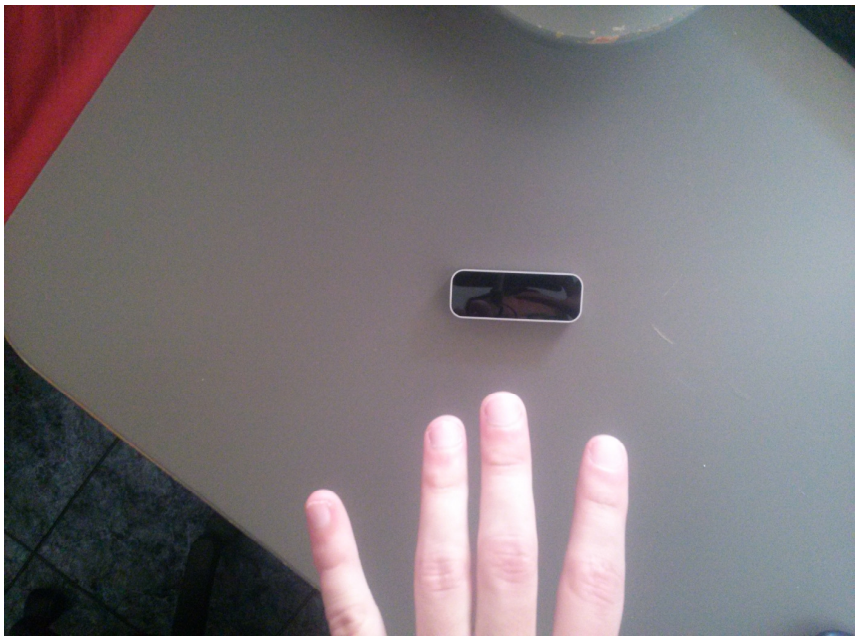

Figura 9.12. Dídac Coll. Imatge que mostra la posició de la mà per fer anar el drone endarrere, 27 de maig del 2015.

Figura 9.13. Dídac Coll. Imatge que mostra la posició de la mà per fer anar el drone cap a l'esquerra. 27 de maig del 2015.

Figura 9.14. Dídac Coll. Imatge que mostra la posició de la mà per fer anar el drone cap a la dreta. 27 de maig del 2015.

Figura 9.15. Dídac Coll. Imatge que mostra la posició de la mà per fer pujar el drone verticalment. 27 de maig del 2015.

Figura 9.16. Dídac Coll. Imatge que mostra la posició de la mà per fer baixar el drone verticalment. 27 de maig de 2015.

Figura 9.17. Dídac Coll. Imatge que mostra la posició de la mà per fer rotar el drone sobre si mateix cap a l'esquerra. 27 de maig de 2015.

Figura 9.18. Dídac Coll. Imatge que mostra la posició de la mà per fer rotar el drone sobre si mateix cap a l'esquerra. 27 de maig de 2015.

9.3 Blocs Simulink Kinect

9.3.1 NID IMAQ

Aquest bloc s'encarrega d'adquirir totes les dades de la Kinect. Entre altres coses també podem controlar l'angle de la Kinect.

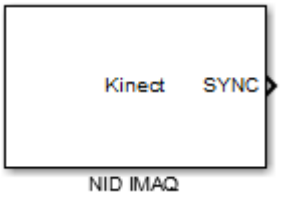

Figura 9.19. Bloc NID IMAQ.

9.3.2 NID Image

Retorna la imatge en RGB de la Kinect

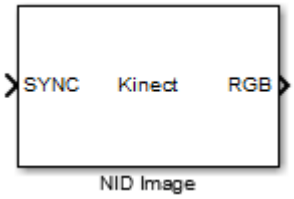

Figura 9.20. Bloc NID Image.

9.3.3 NID Depth

Retorna les dades de profunditat captades per la Kinect.

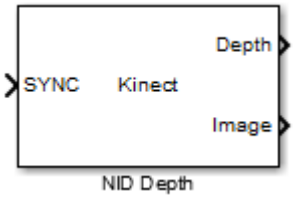

Figura 9.21. Bloc NID Depth.

9.3.4 NID IR

Retorna les dades dels feixos infrarojos captades per la Kinect.

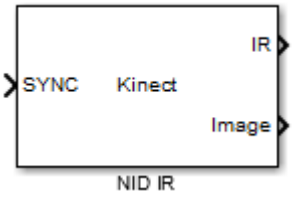

Figura 9.22. Bloc NID IR.

9.3.5 NID Motion

Retorna informació sobre els objectes que s'estan movent.

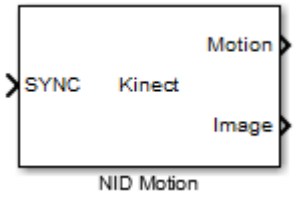

Figura 9.23. Bloc NID Motion.

9.3.6 NID Skeleton

Retorna les dades de les articulacions captades per la Kinect.

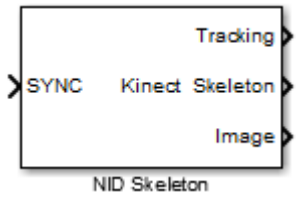

Figura 9.24. Bloc NID Skeleton

9.4 Descripció de la llibreria de comandes Kinect

9.4.1 Videoinput

Comanda per crear un objecte Kinect per poder utilitzar la càmera o el sensor de profunditat.

Implementació

Per crear un objecte Kinect i utilitzar la càmera es pot implementar de la següent manera:

```
videoCamera = videoinput('kinect',1);
```

Per crear un objecte Kinect i utilitzar el sensor es pot implementar de la següent manera:

```
videoDepth = videoinput('kinect',2);
```

9.4.2 Trigger

S'encarrega d'executar "trigger" d'una entrada analògica, és a dir, en aquest cas capta la informació de la Kinect.

Implementació

Inicialment s'ha de configurar l'objecte que s'utilitzarà en aquest cas es mostrarà l'exemple de com utilitzar-lo per captar dades amb el sensor de profunditat.

Primer pas, configurem l'objecte per què agafi un "frame" per cada cop que s'executa el trigger i volem que això s'occeixi infinites vegades:

```
videoDepth.FramesPerTrigger = 1;
videoDepth.TriggerRepeat = inf;
```

Segon pas, configurarem el "trigger" en mode manual. És una recomanació de Matlab en cas que s'utilitzi per adquirir imatges.

```
triggerconfig(videoDepth,'manual');
```

Finalment, ja es pot cridar el "trigger" dintre d'un bucle, per exemple, executem el "trigger".

```
trigger(videoDepth);
```

9.4.3 Getselectedsource

Permet configurar la càmera Kinect segons les nostres necessitats i ens retorna la configuració inicial.

Implementació

En aquest cas s'implementarà un exemple per a configurar el sensor de profunditat i adquirir dades.

```
videpDepthSource = getselectedsource(videodepth)
```

Ens retornaria el següent:

Accelerometer	Retorna un vector 3D de l'acceleració de les dades del sensor de profunditat i color.
BodyPosture	Indica si l'esquelet detectat està assegut o plantat.
CameraElevationAngle	Indica i controla l'angle d'elevació de la càmera Kinect.
DepthMode	Indica el rang de profunditat al mapa de profunditat.
FrameRate	"Frames" per segon per l'adquisició de dades. Sempre està fixat a 30 fps.
IREmitter	Controla si l'emissor d'infrarojos està encès o apagat.
SkeletonsToTrack	Indica la identificació de cada esquelet que ha rastrejat la Kinect.
TrackingMode	Indica el mode de rastreig.

Taula 9.1. Taula descriptiva de les diferents opcions de configuració de la Kinect.

Es recomana consultar el següent enllaç per una informació més detallada: *http://es.mathworks.com/help/imaq/acquiring-image-and-skeletal-data-using-the-kinect.html*

9.4.4 Getdata

S'encarrega d'obtenir les dades de la càmera o del sensor de profunditat. En el cas de la imatge tan sols retorna els "frames", en canvi en el sensor de profunditat retorna els "frames" el temps en què s'adquireix i altres dades com són la posició dels nodes de l'esquelet.

Implementació

```
[frames,time,metaData] = getdata(videoDepth);
```

La següent taula ens il·lustra amb el contingut de la variable metaData:

AbsTime	Representa la marca de temps incloent-t'hi la data amb format del rellotge de Matlab. La variable és de 1x1 doble.
FrameNumber	Una variable 1x1 doble que representa el número de "frames".
IsPositionTracked	És una matriu booleana d'1 x 6 que mostra l'estat de la posició rastrejada de cada un dels sis possibles esquelets que es pot detectar la Kinect. Un 1 indica que s'ha detectat una posició, un 0 indica el contrari.
IsSkeletonTracked	És una matriu booleana d'1 x 6 que mostra l'estat de rastreig de cada un dels 6 esquelets. Un 1 indica que s'ha detectat l'esquelet, un 0 el contrari.

JointDepthIndices	Si la propietat "BodyPosture" està configurada com "Standing", retorna una matriu tipus doble de 20 x 2 x 6 amb coordenades x i y dels 20 nodes amb píxels relatius a la profunditat de la imatge adquirida, sí "BodyPosture està configurada com "Seated" retorna una matriu de 10 x 2 x 6.
JointImageIndices	Si la propietat "BodyPosture" està configurada com "Standing", retorna una matriu tipus doble de 20 x 2 x 6 amb coordenades x i y dels 20 nodes amb píxels relatius a la imatge en colors adquirida, sí "BodyPosture està configurada com "Seated" retorna una matriu de 10 x 2 x 6.
JointTrackingState	És una matriu d'enters de 20 x 6 que conté els valors enumerats per la precisió de cada node rastrejat. 0 indica no rastrejat. 1 indica l'evidència del node. 2 indica que s'ha rastrejat.
JointWorldCoordinates	Retorna una matriu tipus doble de 20 x 3 x 6 de les coordenades x, y i z dels 20 nodes. En cas que "BodyPosture" estigui configurat en mode "Seated" sols retorna 10 nodes.
PositionDepthIndices	Retorna una matriu de tipus doble de 2 x 6 de les coordenades x i y de cada esquelet en pixes relatius a la imatge del sensor de profunditat.
PositionImageIndices	Retorna una matriu de tipus doble de 2 x 6 de les coordenades x i y de cada esquelet en pixes relatius a la imatge del sensor de color.
PositionWorldCoordinates	Retorna una matriu de tipus doble de 3 x 6 de les coordenades x, y i z de cada esquelet en coordenades reals relatius al sensor.

RelativeFrame	És una variable de tipus doble que retorna el número relatiu del "frame" que s'està executant mitjançant el "trigger".
SegmentationData	Retorna el mapa de segmentació. Aquest és un mapa de bits amb valors de píxel corresponent a l'índex de la persona al camp de visió que és més proper a la càmera en aquesta posició del píxel. Un valor de 0 significa que no i esquelet rastrejat.
SkeletonTrackingID	És una matriu d'enter d'1 x 6 que conté la identificació dels sis esquelets.
TriggerIndex	És una variable tipus doble que representa l'esdeveniment de "trigger".

Taula 9.2. Taula descriptiva de les diferents dades calculades per la Kinect.

Per a més informació més es recomana consultar el següent enllaç: *http://es.mathworks.com/help/imaq/acquiring-image-and-skeletal-data-using-the-kinect.html*

9.5 Classe per a connectar-se al AR Drone amb Matlab

Com ja s'havia anomenat a l'apartat de "Connexió entre el Matlab i el AR Drone" de les especificacions tècniques del AR Drone, l'arxiu "AR Drone" proporcionat per Philip Hege[4] ha estat modificat per tal de millorar el control del Drone i a més obtenir les dades de navegació. El codi complet el podeu trobar a l'apartat "Annexes", "Altres": "Classe que mostra les funcions bàsiques del AR Drone"

Un cop vaig començar a realitzar proves amb l'arxiu que proporcionava aquest em vaig adonar compte que apareixien molts errors i no em proporcionava les dades de navegació. En concret l'error que mostrava era el de la figura 8.25.

```
Attempted to access data(25); index out of bounds because numel(data)=24.

Error in ARDrone/ARDrone/navPacketRxCallback (line 70)
                        vBattery = [data(25) data(26) data(27) data(28)];

Error in instrcb (line 36)
        feval(val{1}, obj, eventStruct, val{2:end});

Warning: The BytesAvailableFcn is being disabled. To enable the callback property
either connect to the hardware with FOPEN or set the BytesAvailableFcn property.
```

Figura 9.25. Captura de pantalla dels Errors de l'Arxiu AR Drone

Observant el codi de Philip, podem observar que no realitza un control del buffer si s'ha omplert amb els 500 bytes. Per tant, afegint un condicional que assegura que la variable "data" conté 500 B resolem el problema.

```
if ~isempty(data) && drone.sizeData == 500
```

Un altre problema que també em vaig trobar alhora de controlar el drone és quan li enviava una comanda perquè aquest es mogués. Aquest realitzava l'acció però amb un petit inconvenient, ja que quan deixava d'enviar la comanda aquest continuava fent el mateix moviment. Per tal de què el lector ho comprengui millor ho il·lustrarem amb un exemple: el programa que ens proporcionava Philip, permet controlar el drone amb les tecles de l'ordenador. Per tant, quan es polsava la tecla corresponent a l'acció de fer anar el drone endavant, aquest ho feia. Però malgrat deixar-la de polsar, aquest continuava anant endavant.

Per tal que no deixi d'enviar les dades de navegació s'ha de reiniciar el bit del seu "WhatchDog Timer". El problema del programa de Philip, és el reinici continu que realitza sobre el bit.

Per tant, per solucionar el problema s'ha optat a què el reinici del bit es realitzi si no enviem una comanda en més d'1,2 s [3]. Vegeu el codi:

```
if toc >= 1.2 %Sinó enviem una comanda en menys de 1,2s
                    ... es reinicia el bit
                    AR_WDG
=strcat('AT*COMWDG=',num2str(drone.seq),',');
                    fprintf(drone.ARc, AR_WDG);
                    drone.seq = drone.seq + 1;
                    tic
end
```

Finalment, es pot consultar tot el codi utilitzat per realitzar el control del drone al següent apartat.

9.5.1 Codi del programa

```
classdef ARDrone < handle
    % ARDrone Class
    % Connect AR Drone Parrot and control him
   properties
        seq = 3;
        ARc;
        ARn;
        % Dades de navegació
        Control_State = [];
        Battery_Voltage = 0;
        Pitch = 0;
        Roll = 0;
        Yaw = 0;
        Altitude = 0;
        X_Velocity = 0;
        Y_Velocity = 0;
        NavData = 0;
        sizeData = 0;
        time = tic;
   end
    methods
        function [obj] = ARDrone()
             instrreset
             clc

           %Create Control connection
           obj.ARc = udp('192.168.1.1', 5556, 'LocalPort', 5556);
           fopen(obj.ARc);

           %Create NavData connection
           obj.ARn = udp('192.168.1.1', 5554, 'LocalPort', 5554, ...
            'ByteOrder', 'littleEndian', ...
            'InputBufferSize', 500, ...
            'BytesAvailableFcn', {@navPacketRxCallback,obj}, ...
            'DatagramTerminateMode', 'on', ...
            'BytesAvailableFcnMode', 'byte', ...
            'BytesAvailableFcnCount', 24);
            fopen(obj.ARn);

            % Byte to NavData port for initialization
            fwrite(obj.ARn, 1);
            pause(0.5);
            % NavData command to command port
            AR_NAV_CONFIG =
sprintf('AT*CONFIG=2,\"general:navdata_demo\",\"TRUE\"\r');
            fprintf(obj.ARc, AR_NAV_CONFIG);

            AR_NAV_CTRL = sprintf('AT*CTRL=0');
            fprintf(obj.ARc, AR_NAV_CTRL);

            drone.time=tic; %Comanda per iniciar el contador d'un
timer
         function navPacketRxCallback(obj, event, drone)
```

```
            if (get(obj, 'BytesAvailable') ~=0)
                data = fread(obj,obj.BytesAvailable);
                data = uint8(data);
                drone.sizeData = numel(data);
                %Controlem que data no estigui buida i tingui 500
bytes
                if ~isempty(data) && drone.sizeData == 500

                    % Drone Control State Information
                      drone_state = [data(5) data(6) data(7) data(8)];
                      drone_state = typecast(drone_state, 'uint32');
                      for i = 1:32
                         drone.Control_State(i) = bitget(drone_state,
i);
                      end

                    % Battery Voltage Percentage
                      vBattery = [data(25) data(26) data(27)
data(28)];
                      drone.Battery_Voltage = typecast(vBattery,
'uint32');

                    % Theta (Pitch) Values
                      tPitch = [data(29) data(30) data(31) data(32)];
                      drone.Pitch = typecast(tPitch, 'single') / 1000;

                    % Phi (Roll) Values
                      tRoll = [data(33) data(34) data(35) data(36)];
                      drone.Roll = typecast(tRoll, 'single') / 1000;

                    % Psi (Yaw) Values
                      tYaw = [data(37) data(38) data(39) data(40)];
                      drone.Yaw = typecast(tYaw, 'single') / 1000;

                    % Altitude
                      tAltitude = [data(41) data(42) data(43)
data(44)];
                      drone.Altitude = single(typecast(tAltitude,
'int32'))/ 1000;

                    % X Velocity
                      Vx = [data(45) data(46) data(47) data(48)];
                      drone.X_Velocity = typecast(Vx, 'single') /
1000;

                    % Y Velocity
                      Vy = [data(49) data(50) data(51) data(52)];
                      drone.Y_Velocity = typecast(Vy, 'single') /
1000;

                    % Entire NavData Packet
                      drone.NavData = data;
                end

                % Reinciem el Drone Watchdog Bit
                if toc(drone.time) >= 1.2 %Sinó enviem una comanda en
menys de 1,2s
                    ... es reinicia el bit
```

```
                    AR_WDG =
strcat('AT*COMWDG=',num2str(drone.seq),',');
                    fprintf(drone.ARc, AR_WDG);
                    drone.seq = drone.seq + 1;
                    drone.time=tic;
                end

            end
        end
    end

        function command(obj, command_type, code)
            AR_CMD = sprintf( 'AT*%s=%i,%s\r', command_type, obj.seq,
code);
            fprintf(obj.ARc, AR_CMD);
            obj.seq = obj.seq + 1;
        end

        function takeoff(obj)
            obj.command('FTRIM','')
            obj.command('CONFIG','\"control:altitude_max\",\"20000\"')
            obj.command('REF','290718208')
        end

        function hover(obj)
            obj.drive([0,0,0,0])
        end

        function land(obj)
          obj.command('REF','290717696')
        end

        function emergency(obj)
            obj.command('REF','290717952')
            obj.command('REF','290717696')
        end

        function drive(obj, speed)
            % Controls the Drone movement directly using motor speeds
            fvel = typecast(single(speed(1)/2), 'int32');
            lvel = typecast(single(speed(2)/2), 'int32');
            uvel = typecast(single(speed(3)/2), 'int32');
            rvel = typecast(single(speed(4)/2), 'int32');
            fvel_str = strcat(num2str(fvel),',');
            lvel_str = strcat(num2str(lvel),',');
            uvel_str = strcat(num2str(uvel),',');
            rvel_str = num2str(rvel);
            obj.command('PCMD',strcat('1,',...
                lvel_str,  fvel_str,  uvel_str,  rvel_str))
            obj.time=tic;
        end

        function moveUp(obj, speed)
        obj.drive([0,0,speed,0])
        end

        function moveDown(obj, speed)
         % Moves the Drone Down at a specific speed
        obj.drive([0,0,-speed,0])
        end
```

```
        function moveLeft(obj, speed)
        % Moves the Drone left at a specific speed
        obj.drive([0,-speed,0,0])
        end

        function moveRight(obj, speed)
        % Moves the Drone right at a specific speed
        obj.drive([0,speed,0,0])
        end

        function moveForward(obj, speed)
        % Moves the Drone forward at a specific speed
        obj.drive([-speed,0,0,0])
        end

        function moveReverse(obj, speed)
        % Moves the Drone forward at a specific speed
        obj.drive([speed,0,0,0])
        end

        function rotateLeft(obj, omega)
        % Rotates Drone left at a specific speed
            obj.drive([0,0,0, -omega])
         end

        function rotateRight(obj, omega)
        % Rotates Drone Right at a specific speed
            obj.drive([0,0,0, omega])
        end

        function stop(obj)
        % Stop all Drone motion and close communications.
            obj.land
            fclose(obj.ARn);
            fclose(obj.ARc);
        end

    end
end
```

9.6 Comandes Matlab per l'Arduino

```
%-- connect to the board
 a = arduino('COM9') %Look witch port arduino is connected
 %-- specify pin mode
 a.pinMode(4,'input');
 a.pinMode(13,'output');
 %-- digital i/o
 a.digitalRead(4) % read pin 4
 a.digitalWrite(13,0) % write 0 to pin 13
 %-- analog i/o
 a.analogRead(5) % read analog pin 5
 a.analogWrite(9, 155) % write 155 to analog pin 9
 %-- serial port
 a.serial % get serial port
 a.flush; % flushes PC's input buffer
 a.roundTrip(42) % sends 42 to the arduino and back
 %-- servos
 a.servoAttach(9); % attach servo on pin #9
 a.servoWrite(9,100); % rotates servo on pin #9 to 100 degrees
 val=a.servoRead(9); % reads angle from servo on pin #9
 a.servoDetach(9); % detach servo from pin #9
 %-- encoders
 a.encoderAttach(0,3,2) % attach encoder #0 on pins 3 (pin A) and 2 (pin
B)
 a.encoderRead(0) % read position
 a.encoderReset(0) % reset encoder 0
 a.encoderStatus; % get status of all three encoders
 a.encoderDebounce(0,12) % sets debounce delay to 12 (~1.2ms)
 a.encoderDetach(0); % detach encoder #0
 %-- adafruit motor shield (with AFMotor library)
 a.motorRun(4, 'forward') % run motor forward
 a.stepperStep(1, 'forward', 'double', 100); % move stepper motor
 %-- close session
 delete(a)
```

9.8 Programes Simulink per la Kinect

En aquest apartat es pretén mostrar tot el codi que ha sigut necessari per crear la GUI que executa cada un dels programes Simulink per mostrar els angles de les parts del cos superior. A més a més, s'ha afegit de cada programa els seus respectius blocs utilitzats.

9.8.1 Codi de la GUI creada per als programes Simulink

```
function varargout = Main_menu(varargin)

% MAIN_MENU MATLAB code for Main_menu.fig
%      MAIN_MENU, by itself, creates a new MAIN_MENU or raises the
existing
%      singleton*.
%
%      H = MAIN_MENU returns the handle to a new MAIN_MENU or the
handle to
%      the existing singleton*.
%
%      MAIN_MENU('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in MAIN_MENU.M with the given input
arguments.
%
%      MAIN_MENU('Property','Value',...) creates a new MAIN_MENU or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before Main_menu_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Main_menu_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Main_menu

% Last Modified by GUIDE v2.5 08-Mar-2015 00:00:58

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Main_menu_OpeningFcn, ...
                   'gui_OutputFcn',  @Main_menu_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Main_menu is made visible.



function Main_menu_OpeningFcn(hObject, eventdata, handles, varargin)

% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Main_menu (see VARARGIN)

% Choose default command line output for Main_menu
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Main_menu wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Main_menu_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in EL_Ang.
function EL_Ang_Callback(hObject, eventdata, handles)
% status = get(handles.sim_text, 'String');
open_system('ElbowLeft')
state_button = get(hObject, 'String');
if strcmp(state_button, 'Elbow Left Angle')
     set_param(bdroot,'SimulationCommand','start')
    set(handles.sim_text,'String','Elbow Left Angle');
    set(hObject,'String', 'STOP')
elseif strcmp (state_button, 'STOP')
    set_param(bdroot,'SimulationCommand','stop');
      set(handles.sim_text,'String','None Simulation');
    set(hObject,'String', 'Elbow Left Angle')
    close_system('ElbowLeft')
end

% --- Executes on button press in ShL_Ang.
function ShL_Ang_Callback(hObject, eventdata, handles)
open_system('ShoulderLeft')
state_button = get(hObject, 'String');
if strcmp(state_button, 'Shoulder Left Angle')
     set_param(bdroot,'SimulationCommand','start')
```

```
      set(handles.sim_text,'String','Shoulder Left Angle');
    set(hObject,'String', 'STOP')
elseif strcmp (state_button, 'STOP')
    set_param(bdroot,'SimulationCommand','stop');
     set(handles.sim_text,'String','None Simulation');
    set(hObject,'String', 'Shoulder Left Angle')
    close_system('ShoulderLeft')
end

% hObject    handle to ShL_Ang (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in ER_Ang.
function ER_Ang_Callback(hObject, eventdata, handles)
open_system('ElbowRight')
state_button = get(hObject, 'String');
if strcmp(state_button, 'Elbow Right Angle')
     set_param(bdroot,'SimulationCommand','start')
      set(handles.sim_text,'String','Elbow Right Angle');
    set(hObject,'String', 'STOP')
elseif strcmp (state_button, 'STOP')
    set_param(bdroot,'SimulationCommand','stop');
     set(handles.sim_text,'String','None Simulation');
    set(hObject,'String', 'Elbow Right Angle')
    close_system('ElbowRight')
end
% hObject    handle to ER_Ang (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)

% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1

 % --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

 % --- Executes on button press in ShR_Ang.
function ShR_Ang_Callback(hObject, eventdata, handles)
open_system('ShoulderRight')
```

```
state_button = get(hObject, 'String');
if strcmp(state_button, 'Shoulder Right Angle')
    set_param(bdroot,'SimulationCommand','start')
    set(handles.sim_text,'String','Shoulder Right Angle');
    set(hObject,'String', 'STOP')
elseif strcmp (state_button, 'STOP')
    set_param(bdroot,'SimulationCommand','stop');
    set(hObject,'String', 'Shoulder Right Angle')
     set(handles.sim_text,'String','None Simulation');
    close_system('ShoulderRight')
end
% hObject    handle to ShR_Ang (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 function sim_text_Callback(hObject, eventdata, handles)
% hObject    handle to sim_text (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sim_text as text
%        str2double(get(hObject,'String')) returns contents of
sim_text as a double
set(handles.sim_text,'String', num2str(skeleton))

% --- Executes during object creation, after setting all properties.
function sim_text_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sim_text (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

 % --- Executes on button press in SA.
function SA_Callback(hObject, eventdata, handles)
open_system('Calcul_vectorial_signe')
state_button = get(hObject, 'String');
if strcmp(state_button, 'Simulate All')
    set_param(bdroot,'SimulationCommand','start')
    set(handles.sim_text,'String','Simulate All');
    set(hObject,'String', 'STOP')
elseif strcmp (state_button, 'STOP')
    set_param(bdroot,'SimulationCommand','stop');
    set(hObject,'String', 'Simulate All')
     set(handles.sim_text,'String','None Simulation');
    close_system('Calcul_vectorial_signe')
end
% hObject    handle to SA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 % --- Executes on button press in SLY.
function SLY_Callback(hObject, eventdata, handles)
open_system('ShoulderLeftYawn')
```

```
state_button = get(hObject, 'String');
if strcmp(state_button, 'Shoulder Left Yawn')
    set_param(bdroot,'SimulationCommand','start')
    set(handles.sim_text,'String','Shoulder Left Yawn');
    set(hObject,'String', 'STOP')
elseif strcmp (state_button, 'STOP')
    set_param(bdroot,'SimulationCommand','stop');
    set(hObject,'String', 'Shoulder Left Yawn')
     set(handles.sim_text,'String','None Simulation');
    close_system('ShoulderLeftYawn')
end
% hObject    handle to SLY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in SRY.
function SRY_Callback(hObject, eventdata, handles)
open_system('ShoulderRightYawn')
state_button = get(hObject, 'String');
if strcmp(state_button, 'Shoulder Right Yawn')
    set_param(bdroot,'SimulationCommand','start')
    set(handles.sim_text,'String','Shoulder Right Yawn');
    set(hObject,'String', 'STOP')
elseif strcmp (state_button, 'STOP')
    set_param(bdroot,'SimulationCommand','stop');
    set(hObject,'String', 'Shoulder Right Yawn')
     set(handles.sim_text,'String','None Simulation');
    close_system('ShoulderRightYawn')
end
% hObject    handle to SRY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

9.8.2 Elbow Left Angle

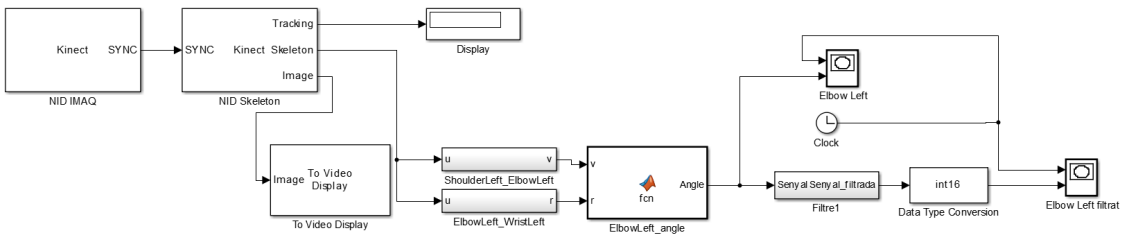

Figura 9.26. Blocs que conté l'arxiu Simulink "Elbow Left Angle".

9.8.2.1 ShoulderLeft_ElbowLeft

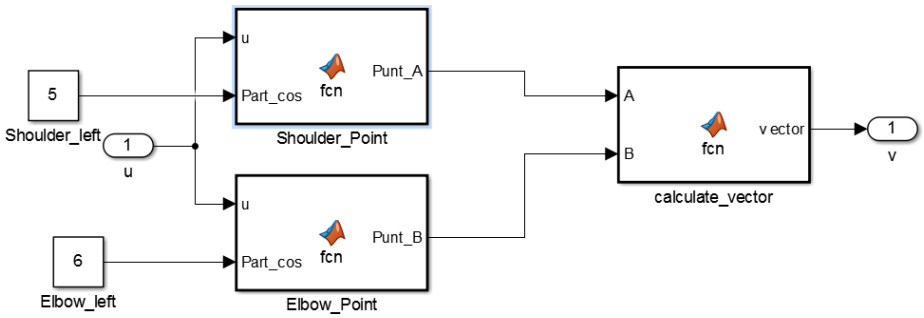

Figura 9.27. Blocs que conté el bloc "ShoulderLeft_ElbowLeft".

Shoulder_Point

```
function Punt_A  = fcn(u,Part_cos)
%#eml
x=u(:,Part_cos,1);
y=u(:,Part_cos,2);
z=u(:,Part_cos,3);
Punt_A = [x y z];
```

Elbow_Point

```
function Punt_B  = fcn(u,Part_cos)
%#eml
x=u(:,Part_cos,1);
y=u(:,Part_cos,2);
z=u(:,Part_cos,3);
Punt_B = [x y z];
```

calculate_Vector

```
function vector  = fcn(A,B)
%#eml
vector = B - A;
```

9.8.2.2 ElbowLeft_WristLeft

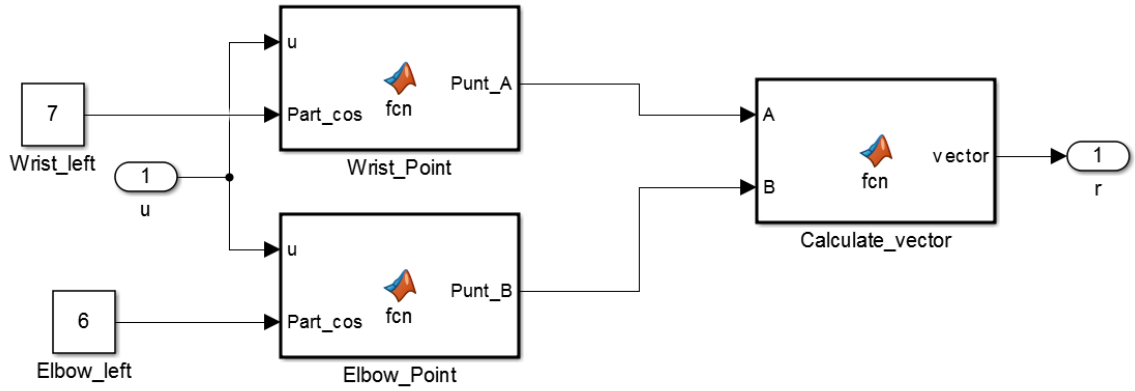

Figura 9.28. Blocs que conté el bloc "ElbowLeft_WristLeft".

Wrist_Point

```
function Punt_A  = fcn(u,Part_cos)
%#eml
```

```
x=u(:,Part_cos,1);
y=u(:,Part_cos,2);
z=u(:,Part_cos,3);
Punt_A = [x y z];
```

Elbow_Point

```
function Punt_B  = fcn(u,Part_cos)
%#eml
x=u(:,Part_cos,1);
y=u(:,Part_cos,2);
z=u(:,Part_cos,3);
Punt_B = [x y z];
```

Calculate_vector

```
function vector  = fcn(A,B)
%#eml
vector = B - A;
```

9.8.2.3 ElbowLeft_angle

```
function Angle  = fcn(v,r)
%#eml
Producte_escalar = dot(v,r); %Calcul producte escalar
Angle_radians= acos(Producte_escalar/(norm(v,2)*norm(r,2)));
Angle = Angle_radians*180/ pi;
```

9.8.2.4 Filtre1

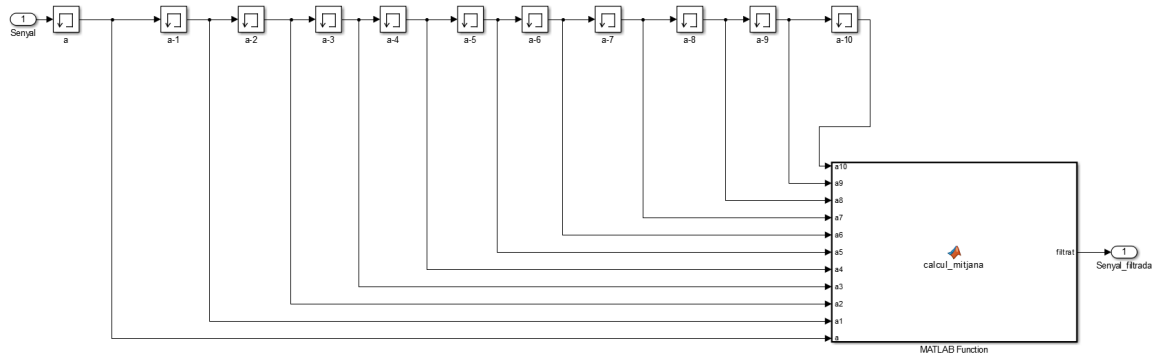

Figura 9.29. Blocs que conté el bloc "Filtre1".

Càlcul_mitjana

```
function filtrat  = calcul_mitjana(a10,a9,a8,a7,a6,a5,a4,a3,a2,a1,a)
%#eml
filtrat= (a+a1+a2+a3+a4+a5+a6+a7+a8+a9+a10)/10;
```

9.8.3 Elbow Right Angle

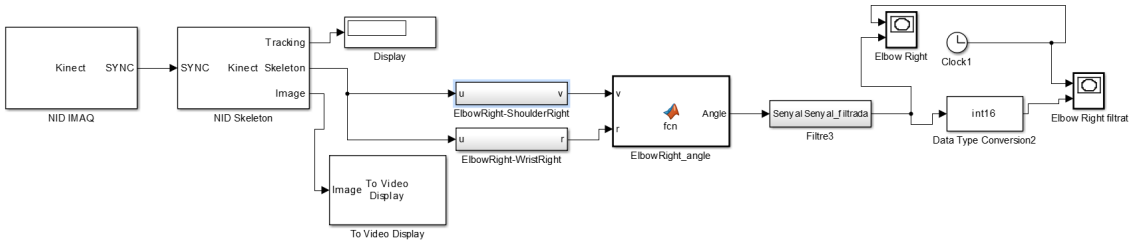

Figura 9.30. Blocs que conté l'arxiu simulink "Elbow Right Angle".

9.8.4 ElbowRight-ShoulderRight

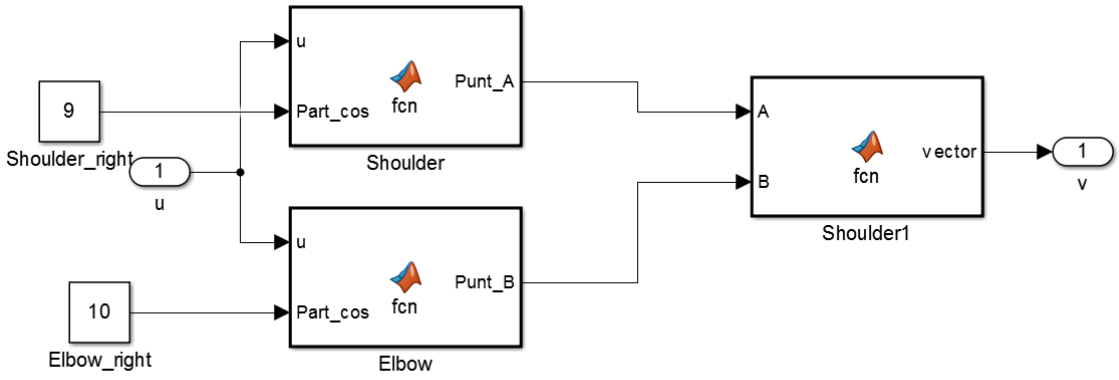

Figura 9.31. Blocs que conté el bloc "ElbowRight-ShoulderRight".

9.8.4.1 ElbowRight-WristRight

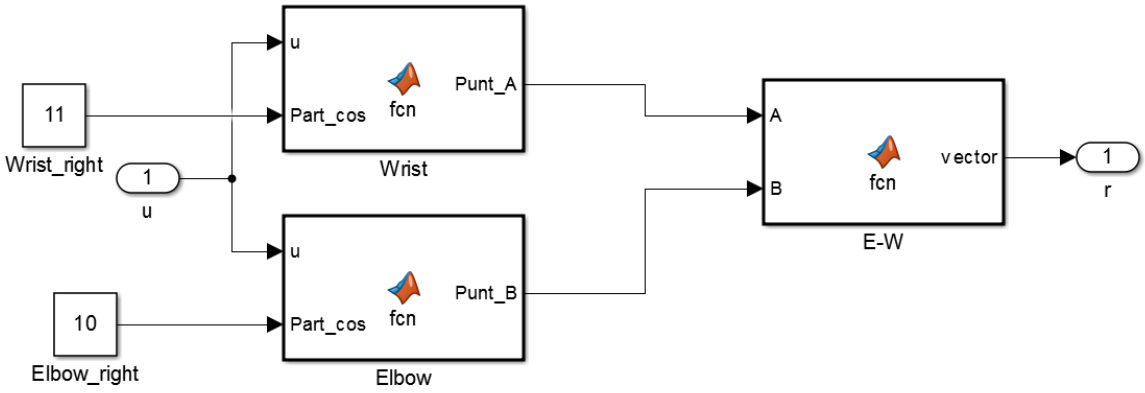

Figura 9.32. Blocs que conté el bloc "ElbowRIght-WristRight".

Com que la resta de codi és idèntic al descrit a l'apartat "Elbow Left Angle" no es donarà més detalls sobre el contingut dels blocs.

9.8.5 Shoulder Left Angle

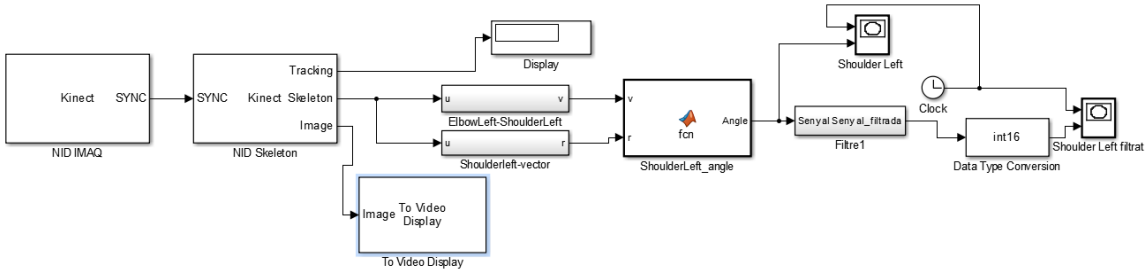

Figura 9.33. Blocs que conté l'arxiu simulink "Shoulder Left Angle".

9.8.5.1 ElbowLeft-ShoulderLeft

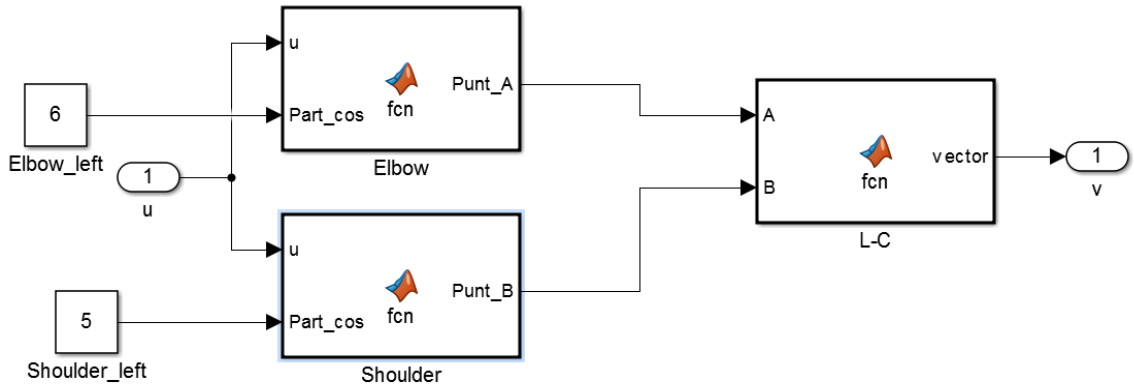

Figura 9.34. Blocs que conté el bloc "ElbowLeft-ShoulderLeft".

9.8.5.2 ShoulderLeft-vector

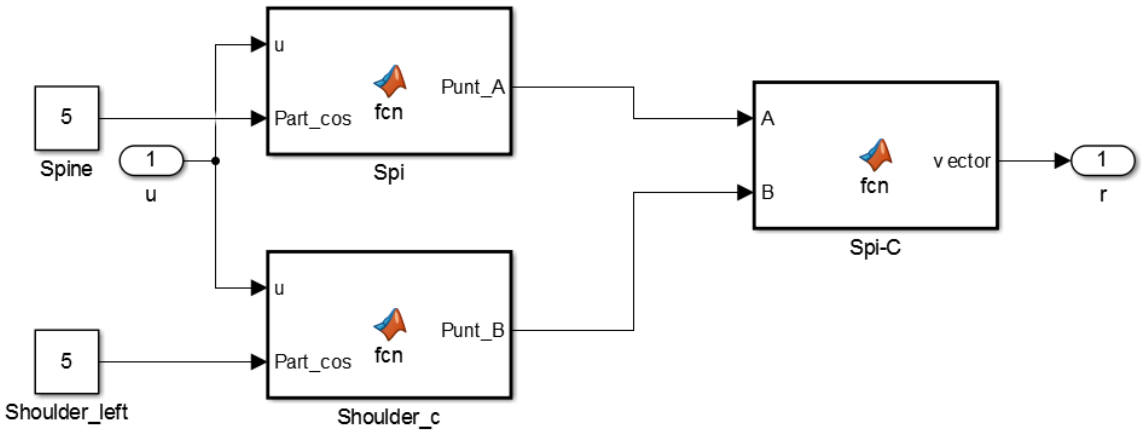

Figura 9.35. Blocs que conté el bloc "ShoulderLeft-vector".

Spi

```
function Punt_A  = fcn(u,Part_cos)
%#eml
x=u(:,Part_cos,1);
y=u(:,Part_cos,2);
z=u(:,Part_cos,3);
B = [x y z];
```

```
vector_unitari = [0 1 0];
Punt_A = B - vector_unitari;
```

Shoulder_c

```
function Punt_B  = fcn(u,Part_cos)
%#eml
x=u(:,Part_cos,1);
y=u(:,Part_cos,2);
z=u(:,Part_cos,3);
Punt_B = [x y z];
```

Spi-C

```
function vector  = fcn(A,B)
%#eml
vector = B - A;
```

Com que la resta de codi és idèntic al descrit a l'apartat "Elbow Left Angle" no es donarà més detalls sobre el contingut dels blocs.

9.8.6 Shoulder Right Angle

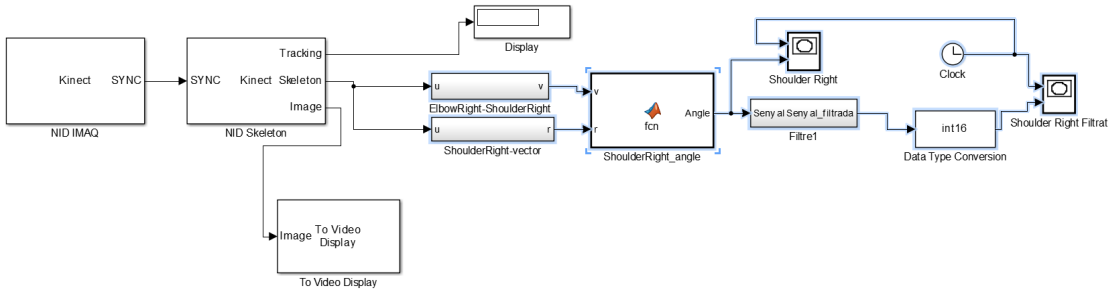

Figura 9.36. Blocs que conté l'arxiu simulink "Shoulder Right Angle".

9.8.6.1 ElbowRight-ShoulderRight

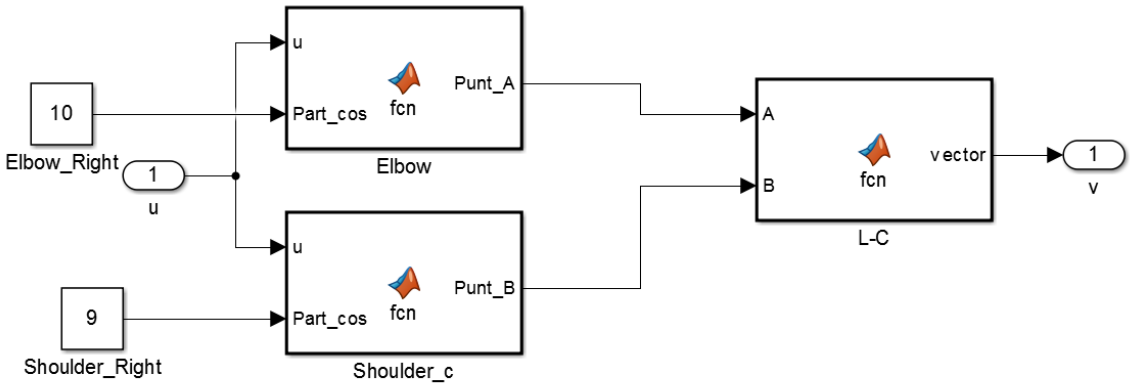

Figura 9.37. Blocs que conté el bloc "ElbowRIght-SHoulderRight".

9.8.6.2 ShoulderRight-vector

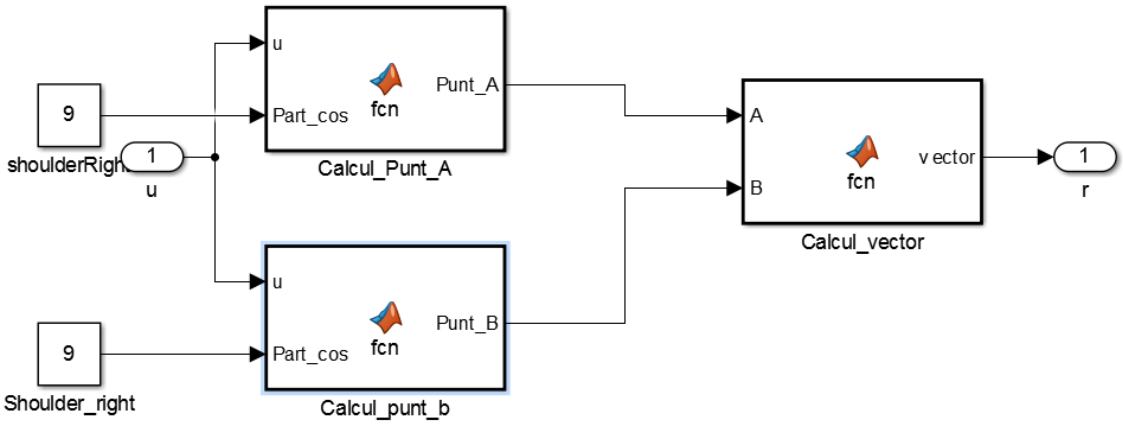

Figura 9.38. Blocs que conté el bloc "ShoulderRIght-vector".

Calcul_Punt_A

```
function Punt_A  = fcn(u,Part_cos)
%#eml
x=u(:,Part_cos,1);
y=u(:,Part_cos,2);
z=u(:,Part_cos,3);
B = [x y z];
vector_unitari = [0 1 0];
Punt_A = B - vector_unitari;
```

Calcul_punt_b

```
function Punt_B  = fcn(u,Part_cos)
%#eml
x=u(:,Part_cos,1);
y=u(:,Part_cos,2);
z=u(:,Part_cos,3);
Punt_B = [x y z];
```

Com que la resta de codi és idèntic al descrit a l'apartat "Elbow Left Angle" no es donarà més detalls sobre el contingut dels blocs.

9.8.7 Shoulder Left Yaw

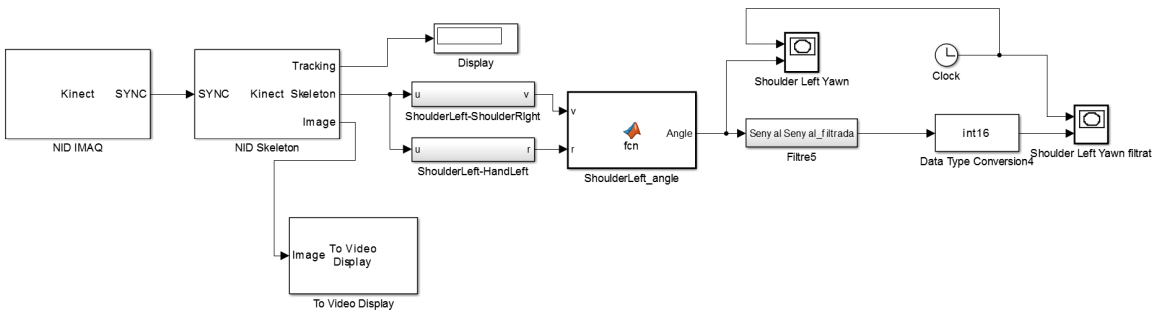

Figura 9.39. Blocs que conté l'arxiu simulink "Shoulder Left Yaw".

Com que la resta de codi és idèntic al descrit a l'apartat "Elbow Left Angle" no es donarà més detalls sobre el contingut dels blocs.

9.8.8 Shoulder Right Yaw

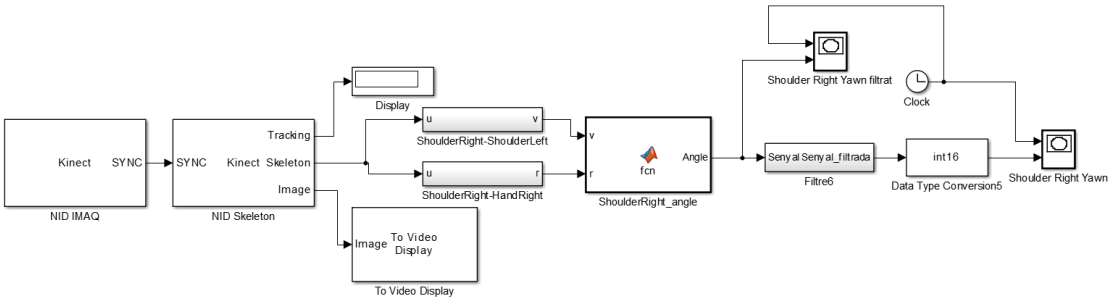

Figura 9.40. Blocs que conté l'arxiu simulink "Shoulder Right Yaw".

Com que la resta de codi és idèntic al descrit a l'apartat "Elbow Left Angle" no es donarà més detalls sobre el contingut dels blocs.

9.8.9 Simulate All

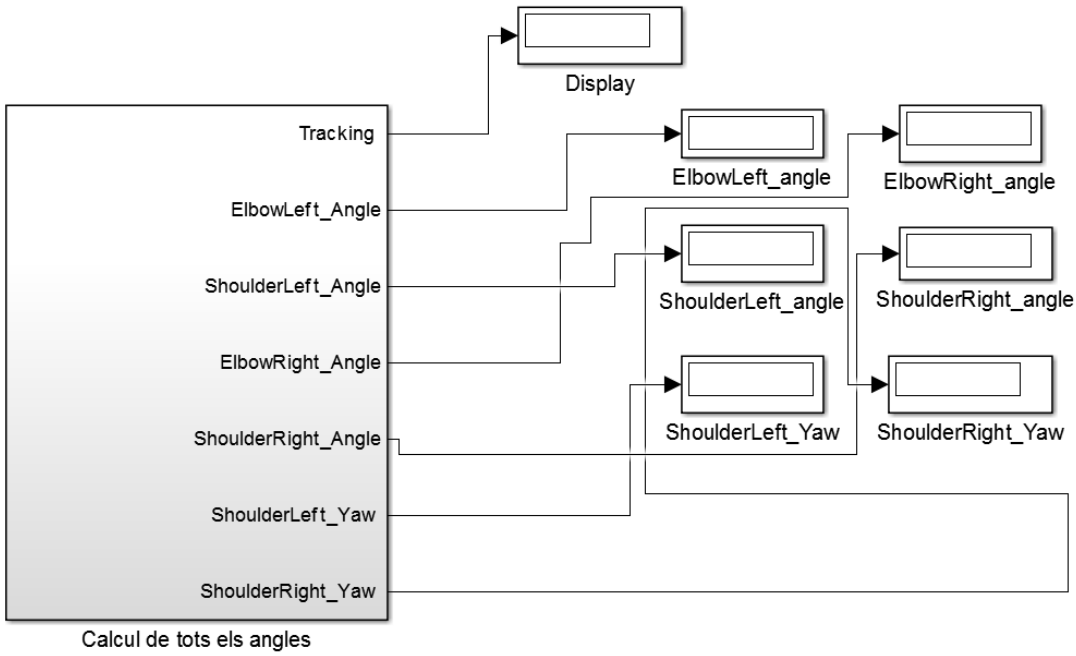

Figura 9.41. Blocs que conté l'arxiu simulink "Simulate All".

9.8.9.1 Càlcul de tots els angles

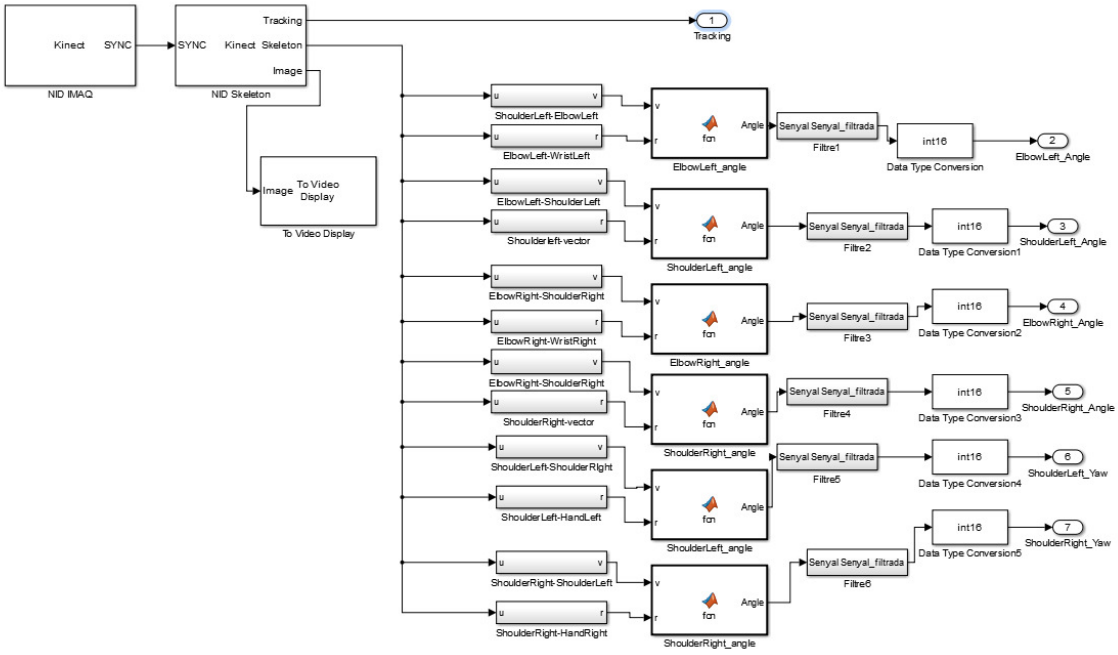

Figura 9.42. Blocs que conté el bloc "Calcul de tots els angles".

El codi que contenen els blocs anteriors és el mateix que els que hem descrit a la resta d'apartats.

9.9 Scripts Kinect

9.9.1 Script per mostrar imatges per pantalla

```
rgbVideo = videoinput('kinect',1) %Creació d'un objecte videoinput per
%mostrar les imatges de video
depthVideo = videoinput('kinect',2) %Creació d'un objecte videoinput
per
%mostrar les imatges i dades del sensor de profunditat

start([rgbVideo depthVid]);

preview(rgbVideo); % Mostra les imatges RGB
preview(depthVideo); % Mostra les imatges del sensor de profunditat
```

9.9.2 Script per a capturar dades de l'esquelet

```
depthVideo = videoinput('kinect',2); %Creació d'un objecte videoinput
%per mostrar les imatges i dades del sensor de profunditat

triggerconfig([rgbVideo,depthVideo],'manual');
depthVideo.FramesPerTrigger = 100; % We're going to take 100 frames
depthSource = getselectedsource(depthVideo); %Get the vídeo source
%object from the depth device's videoinput object.
depthSource.CameraElevationAngle = 0; %We change Kinect's angle
elevation
depthSource.BodyPosture='Standing'; %Standing calculete 20 points
depthSource.TrackingMode ='Skeleton';

start(depthVideo);
trigger(depthVideo); %Begin to take data
% Retrieve the frames and check if any Skeletons are tracked
[frameDataDepth, timeDataDepth, DataDepth] = getdata(depthVideo);
stop(depthVideo);
```

9.9.3 Script per a mostrar l'esquelet en 2D

```
depthVideo = videoinput('kinect',2); %Creació d'un objecte videoinput
per
%mostrar les imatges i dades del sensor de profunditat

triggerconfig(depthVideo,'manual');
depthVideo.FramesPerTrigger = 1; % We're going to take 100 frames
depthVideo.TriggerRepeat = inf;
depthSource = getselectedsource(depthVideo); %Get the videosource
%object from the depth device's videoinput object.
depthSource.CameraElevationAngle = 0; %We change Kinect's angle
elevation
depthSource.BodyPosture='Standing'; %Standing calculete 20 points
depthSource.TrackingMode ='Skeleton';

start(depthVideo);

f = figure;
axis([-1 1 -1 1]);
xlabel('X(m)');
ylabel('Y(m)');
grid on;
while ishandle(f)
```

```
    trigger(depthVideo); %Begin to take data
    %Take Frames
    [frameDataDepth, timeDataDepth, DataDepth] = getdata(depthVideo);
    SkeletonTracked = DataDepth.IsSkeletonTracked;
    whatSkeletonIsTracked = find(DataDepth.IsSkeletonTracked);
    if sum(SkeletonTracked)>0   %Check if one of the six skeletons
are...
        ...tracked
      %Plot the dataDepth in real coordinates.

plot(DataDepth.JointWorldCoordinates(:,1,whatSkeletonIsTracked),...

DataDepth.JointWorldCoordinates(:,2,whatSkeletonIsTracked),'+'),...
        axis([-1 1 -1 1]);
        xlabel('X(m)');
        ylabel('Y(m)');
        grid on;
        drawnow; % upload data
    end
end
stop(depthVideo);
```

9.9.4 Script per a mostrar l'esquelet en un gràfic 3D

```
depthVideo = videoinput('kinect',2); %Creació d'un objecte videoinput
per
%mostrar les imatges i dades del sensor de profunditat

triggerconfig(depthVideo,'manual');
depthVideo.FramesPerTrigger = 1; % We're going to take 100 frames
depthVideo.TriggerRepeat = inf;
depthSource = getselectedsource(depthVideo); %Get the videosource
%object from the depth device's videoinput object.
depthSource.CameraElevationAngle = 0; %We change Kinect's angle
elevation
depthSource.BodyPosture='Standing'; %Standing calculete 20 points
depthSource.TrackingMode ='Skeleton';

start(depthVideo);

f = figure;
axis([-2 2 -0.5 2.5 -1 1]);
xlabel('X(m)');
ylabel('Y(m)');
zlabel('Z(m)');
grid on;
while ishandle(f)
    trigger(depthVideo); %Begin to take data
    % Retrieve the frames and check if any Skeletons are tracked
    [frameDataDepth, timeDataDepth, DataDepth] = getdata(depthVideo);
    SkeletonTracked = DataDepth.IsSkeletonTracked~=0;
    whatSkeletonIsTracked = find(DataDepth.IsSkeletonTracked);
    if sum(SkeletonTracked)>0   %Check if one of the six skeletons
are...
        ...tracked
        %Plot the dataDepth in real coordinates.

plot3(DataDepth.JointWorldCoordinates(:,1,whatSkeletonIsTracked),...

DataDepth.JointWorldCoordinates(:,3,whatSkeletonIsTracked),...
```

```
DataDepth.JointWorldCoordinates(:,2,whatSkeletonIsTracked),...
            '+r','LineWidth',2),axis([-2 2 -0.5 3 -1 1]);
            xlabel('X(m)');
            ylabel('Y(m)');
            zlabel('Z(m)');
            grid on;
            drawnow;
    end
end
stop(depthVideo);
```

9.10 Programa principal de la Kinect

9.10.1 Codi de la GUI del programa principal

```
function varargout = main_kinect(varargin)
% MAIN_KINECT MATLAB code for main_kinect.fig
%  MAIN_KINECT, by itself, creates a new MAIN_KINECT or raises the
existing
%  singleton*.
%
%  H = MAIN_KINECT returns the handle to a new MAIN_KINECT or the handle
to
%  the existing singleton*.
%
%  MAIN_KINECT('CALLBACK',hObject,eventData,handles,...) calls the local
%  function named CALLBACK in MAIN_KINECT.M with the given input
arguments.
%
%  MAIN_KINECT('Property','Value',...)
%  creates a new MAIN_KINECT or raises the
%  existing singleton*.  Starting from the left, property value pairs
are
%  applied to the GUI before main_kinect_OpeningFcn gets called.  An
%  unrecognized property name or invalid value makes property
application
%  stop.  All inputs are passed to main_kinect_OpeningFcn via varargin.
%
%  *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%  instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help main_kinect

% Last Modified by GUIDE v2.5 12-May-2015 01:24:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @main_kinect_OpeningFcn, ...
                   'gui_OutputFcn',  @main_kinect_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before main_kinect is made visible.
function main_kinect_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to main_kinect (see VARARGIN)

% Choose default command line output for main_kinect
% -- Inicialitzacio de handles
handles.output = hObject;
handles.depth = videoinput('kinect',2); % Crea un objecte videoinput
Kinect
...de profunditat
handles.depthSrc = getselectedsource(handles.depth); % Assignemuna
...estructura de dades per accedir a la configuracio de la kinect

handles.botoEstatConfiguracio = 0; % COmproba si s'ha polsat el boto
...configuració configurem el trigger
triggerconfig(handles.depth,'manual');
handles.depth.FramesPerTrigger = 1;
handles.depth.TriggerRepeat = inf;
% Comproba els estats dels botons connectar iagafar dades
handles.stateButtonConnectar = get(handles.buttonConnectar,'String');
%Guardem l'estat del boto connectar
handles.stateAgafarDades = get(handles.buttonAgafarDades,'String');

%Creem variables que estiran al workspace
assignin('base','angleCamera', 0);
assignin('base','bodyPosture','Seated');
assignin('base','trackingMode','Off');

% Craci d'una variable per conneixer si el drone està en l'airat.
handles.takeoff = 0;
% Update handles structure
guidata(hObject, handles);

 % --- Outputs from this function are returned to the command line.
function varargout = main_kinect_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

 function mostrarMissatges_Callback(hObject, eventdata, handles)
% hObject    handle to mostrarMissatges (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function mostrarMissatges_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mostrarMissatges (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end

% --- Executes on button press in buttonConnectar.
function buttonConnectar_Callback(hObject, eventdata, handles)
% hObject    handle to buttonConnectar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 uiresume(handles.figure1);
 if handles.botoEstatConfiguracio
      set(handles.mostrarMissatges,'String','');
   if strcmp(handles.stateButtonConnectar,'Connectar')
      set(hObject,'String','Desconnectar');
      handles.stateButtonConnectar = 'Desconnectar';
      % Agafem el valor de la configuracio
      handles.depthSrc.CameraElevationAngle =
evalin('base','angleCamera');
      % Agafem el tipus de postura que hem configurat
      handles.depthSrc.BodyPosture = evalin('base','bodyPosture');
      handles.depthSrc.TrackingMode = evalin('base','trackingMode');
      % Mostrem pel workspace com ha quedat la configuracio
      handles.depthSrc
   else
      set(hObject,'String','Connectar');
      handles.stateButtonConnectar = 'Connectar';
   end
 else
     set(handles.mostrarMissatges,'String','Error: Configura la
Kinect');
 end
   guidata(hObject,handles);


% --- Executes on button press in buttonAgafarDades.
function buttonAgafarDades_Callback(hObject, eventdata, handles)

global stateButton elAngle erAngle shlAngle shrAngle shlYawAngle...
    shrYawAngle i takenoff vel ang dr
   handles.stateAgafarDades = get(hObject,'String'); % Comproba l'estat
del
   ...boto stop si esta en modo STOP el programa executara el while
   guidata(hObject,handles); % Comproba l'estat del estring agafar dades
   stateButton = handles.stateAgafarDades;
 if strcmp(handles.stateButtonConnectar,'Desconnectar')
   if strcmp( stateButton ,'Agafar Dades')
        set(handles.mostrarMissatges,'String','');
        set(hObject,'String','STOP');
          dr = ARDrone;
        start(handles.depth);
   else
        set(hObject,'String','Agafar Dades');
        delete dr;
   end
 else
   set(handles.mostrarMissatges,'String','Error: Clica al boto
Connectar');
 end

   handles.stateAgafarDades = get(hObject,'String'); % Comproba l'estat
   ...del boto stop si esta en modo STOP el programa executara el while
   guidata(hObject,handles); % Actualitzem la GUI
```

```
   stateButton =  handles.stateAgafarDades;
   i = 1;
   takenoff = 0;
   vel = 0.3; % Velcoitat del drone sobre el pla horitzontal
   ang = 1; % velocitat de rotació i moviment sobre el pla vertical
     %--Inicialisation
    timeArray(1) = 0;
   frameRate=tic;
   displayTime = tic;
   timeDisplayNavData = tic;
 while  strcmp(stateButton,'STOP')
     trigger(handles.depth);
    [~,timeData,metaData]=getdata(handles.depth); % Adquirim les dades
    ...de la kinect
      % Variables per a calcular els frame rate de matlab
        timeArray(2)=timeData;
        time = (timeArray(2)-timeArray(1));
        FrameRate = round(1/time);
        timeArray(1)=timeArray(2);

        if toc(frameRate)>=0.2 %Mostrem el Frame Rate cada 0.2 s
             %Kinect Frame Rate
             set(handles.frameRate,'String',...
                 [num2str(round(handles.depthSrc.FrameRate)) '/30
fps']);
             %Matlab Frame Rate
             set(handles.textFrameRateMtb,'String',...
                 [num2str(round(FrameRate)) ' fps']);
            frameRate=tic;
        end

    if sum(metaData.IsSkeletonTracked) > 0 % Comproba si detecta
l'esquelet
        realCoordSkeleton= metaData.JointWorldCoordinates(:,:,...
            metaData.IsSkeletonTracked); % Llegim les dades en...
        ...coordenades reals
       %--Angle colze esquerre
             elAngle(i)=angleColzeEsquerre(realCoordSkeleton);
             elEnter = int16(elAngle);
             eL = elEnter(i);
       %--Angle colze dret
             erAngle(i) = angleColzeDret(realCoordSkeleton);
             erEnter = int16(erAngle);
             eR = erEnter(i);
       %--Angle balanceig hombro esquerre
             shlAngle(i) = angleHombroEsquerre(realCoordSkeleton);
             shlEnter = int16(shlAngle);
             shl = shlEnter(i);
       %--Angle balanceig hombro dret
             shrAngle(i) = angleHombroDret(realCoordSkeleton);
             shrEnter = int16(shrAngle);
             shr = shrEnter(i);
       %--Angle guinyada hombro esquerre
             shlYawAngle(i) = angleYawHombroEsquerre(realCoordSkeleton);
             shlYawEnter = int16(shlYawAngle);
             shlY = shlYawEnter(i);
       %--Angle guinyada hombro esquerre
             shrYawAngle(i) = angleYawHombroDret(realCoordSkeleton);
             shrYawEnter = int16(shrYawAngle);
             shrY = shrYawEnter(i);
                if handles.takeoff == 0 &&...
```

```
strcmp(get(handles.buttonConnectarDrone,'String'),...
                        'Desconnectar')% Si no ha despegat que ho fagi
                    dr.takeoff(); % Fer despegar el drone
                     handles.takeoff = 1;
                end
                if handles.takeoff == 1
                  if (shl>=80 && shl<=110 && shr<20) % Comprovem si
                  ...el angle del hombro esquerre amb la vertical fa
                  ...aproximadament 90 graus i el hombro dret està en
repòs
                     dr.rotateLeft(ang); %Fem rotar el dron a l'esquerra
                  end
                  if (shr>=80 && shr<=110 && shl<20) %Comprovem si
                  ...el angle del hombro dret amb la vertical fa
                  ...aproximadament 90 graus i el hombro esquerre està
                  ...en repòs
                      dr.rotateRight(ang); %Fem rotar el dron a la dreta
                  end
                  if (shl>=130 && shr>=130) % Si els braços estan
aixecats
                      ...cap d'alt el drone Puja
                     dr.moveUp(ang);
                  end
                  if ( shl >= 80 && shl <= 110 )&&...
                          (shr>=80 && shr<=110)&&(shlY<=30 && shrY<=30)
                       % Si els braços estan a 90 graus el drone baixa
                       dr.moveDown(ang);
                  end
                  if (shl>=65 && shl<=110) && (shr >= 65 && shr <= 110
)...
                          && ( shlY >= 80 && shrY >= 80) % Si els braços
                       ...estan estirats cap endavant
                     dr.moveForward(vel); % El drone es mou cap endavant
                  end
                  if (shl<=50)&&(shr<=50)&&(shlY>=80 && shrY>=80)&&...
                          (eL<=50 &&  eR<=50)% Si els braços estan
arronsats
                       ...el drone es mou cap endarrere
                     dr.moveReverse(vel);
                  end
                  if (shl>=70 && shl<=110) && (shr>=70 && shr<=110)...
                          && (eL<=100) && (eR>=120)%Si el braç dret està
                       ...estirat i el colze esquerre forma un angle més
                       ...petit de 100° movem cap a la dreta
                     dr.moveRight(vel);
                  end
                  if (shl>=70 && shl<=110) && (shr>=70 && shr<=110)...
                          && (eR<=100) && (eL>=120)
                     dr.moveLeft(vel);
                  end
                end
            if  toc(displayTime)>=0.5
                    %--Angle colze esquerre
                        set(handles.elbowLeft,'String',num2str(eL));
                    %--Angle colze dret
                        set(handles.elbowRight,'String',num2str(eR));
                  %--Angle guinyada hombro esquerre
                        set(handles.shoulderLeft,'String',num2str(shl));
                  %--Angle guinyada hombro dret
```

```
set(handles.shoulderRight,'String',num2str(shr));
                    %--Angle capcineig hombro esquerre
                         set(handles.yawLeft,'String',num2str(shlY));
                    %--Angle capcineig hombro esquerre
                         set(handles.yawRight,'String',num2str(shrY));
              displayTime=tic;
           end
    elseif handles.takeoff == 1
        dr.land(); % Aterrar el Drone
        handles.takeoff = 0;
    end
     if
strcmp(get(handles.buttonConnectarDrone,'String'),'Desconnectar')
                  if toc (timeDisplayNavData) >=0.5
                     set(handles.textBateria,'String',...
                        [num2str(dr.Battery_Voltage) '%']);
                     set(handles.textPitch,'String',...
                        [num2str(round(dr.Pitch)) '°']);
                     set(handles.textRoll,'String',...
                        [num2str(round(dr.Roll)) '°']);
                     set(handles.textYaw,'String',...
                        [num2str(round(dr.Yaw)) '°']);
                     set(handles.textAltitud,'String',...
                        [num2str(round(dr.Altitude,3)) ' m']);
                     set(handles.textVx,'String',...
                        [num2str(round(dr.X_Velocity,4)) ' m/s']);
                     set(handles.textVy,'String',...
                        [num2str(round(dr.Y_Velocity,4)) ' m/s']);
                    timeDisplayNavData=tic;
                 end
     end
        i = i + 1;
 end
 if handles.takeoff == 1
     dr.land();
     handles.takeoff=0;
 end
  stop(handles.depth);

% --- Executes on button press in buttonPlotEsquelet.
function buttonPlotEsquelet_Callback(hObject, eventdata, handles)
% hObject    handle to buttonPlotEsquelet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in buttonConfiguracio.
function buttonConfiguracio_Callback(hObject, eventdata, handles)
% hObject    handle to buttonConfiguracio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
connectionState = get(handles.buttonConnectar,'String');
    if strcmp(connectionState,  'Connectar')
        handles.botoEstatConfiguracio = 1;
        run('configuracio_kinect')
    else
        %Control d'error sinó s'ha polsat al boto desconnectar
        set(handles.mostrarMissatges,'String',...
            'Error: Clica al boto Desconnectar per configurar la
Kinect');
    end
```

```
    guidata(hObject,handles);

function elbowLeft_Callback(hObject, eventdata, handles)
% hObject    handle to elbowLeft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 % --- Executes during object creation, after setting all properties.
function elbowLeft_CreateFcn(hObject, eventdata, handles)
% hObject    handle to elbowLeft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
 % Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

 function elbowRight_Callback(hObject, eventdata, handles)
% hObject    handle to elbowRight (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 % --- Executes during object creation, after setting all properties.
function elbowRight_CreateFcn(hObject, eventdata, handles)
% hObject    handle to elbowRight (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

 function shoulderLeft_Callback(hObject, eventdata, handles)
% hObject    handle to shoulderLeft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 % --- Executes during object creation, after setting all properties.
function shoulderLeft_CreateFcn(hObject, eventdata, handles)
% hObject    handle to shoulderLeft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function shoulderRight_Callback(hObject, eventdata, handles)
% hObject    handle to shoulderRight (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function shoulderRight_CreateFcn(hObject, eventdata, handles)
% hObject    handle to shoulderRight (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function yawLeft_Callback(hObject, eventdata, handles)
% hObject    handle to yawLeft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function yawLeft_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yawLeft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function yawRight_Callback(hObject, eventdata, handles)
% hObject    handle to yawRight (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function yawRight_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yawRight (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

 % --- Executes during object creation, after setting all properties.
function frameRate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to frameRate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Executes during object creation, after setting all properties.
function textFrameRateMtb_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textFrameRateMtb (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Executes during object creation, after setting all properties.
function textBateria_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textBateria (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Executes during object creation, after setting all properties.
function textPitch_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textPitch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Executes during object creation, after setting all properties.
function textRoll_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textRoll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Executes during object creation, after setting all properties.
function textYaw_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textYaw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Executes during object creation, after setting all properties.
```

```
function textAltitud_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textAltitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Executes during object creation, after setting all properties.
function textVx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textVx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Executes during object creation, after setting all properties.
function textVy_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textVy (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Executes on button press in buttonConnectarDrone.
function buttonConnectarDrone_Callback(hObject, eventdata, handles)
% hObject    handle to buttonConnectarDrone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if strcmp(get(hObject,'String'),'Connectar Drone')
 set(hObject,'String','Desconnectar');
else
    set(hObject,'String','Connectar Drone');
end
```

9.10.2 GUI per a configurar la Kinect

```
function varargout = configuracio_kinect(varargin)
% CONFIGURACIO_KINECT MATLAB code for configuracio_kinect.fig
%  CONFIGURACIO_KINECT, by itself, creates a new CONFIGURACIO_KINECT or
raises the existing
%      singleton*.
%
%      H = CONFIGURACIO_KINECT returns the handle to a new
CONFIGURACIO_KINECT or the handle to
%      the existing singleton*.
%
%      CONFIGURACIO_KINECT('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in CONFIGURACIO_KINECT.M with the given
input arguments.
%
%      CONFIGURACIO_KINECT('Property','Value',...) creates a new
CONFIGURACIO_KINECT or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before configuracio_kinect_OpeningFcn gets
called.  An
```

```
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to configuracio_kinect_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help configuracio_kinect

% Last Modified by GUIDE v2.5 21-Mar-2015 02:41:11

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @configuracio_kinect_OpeningFcn,
...
                   'gui_OutputFcn',  @configuracio_kinect_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before configuracio_kinect is made visible.
function configuracio_kinect_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to configuracio_kinect (see
VARARGIN)
% Choose default command line output for configuracio_kinect
handles.output = hObject;
% Recupera l'últim estat configurat
set(handles.editAngle,'String',evalin('base','angleCamera'));
switch evalin('base','bodyPosture')
    case 'Standing'
        bnum = 1;
    case 'Seated'
       bnum = 2;
end
set(handles.popupmenu1,'Value',bnum);

switch evalin('base','trackingMode')
    case 'Off'
        tnum = 1;
    case 'Skeleton'
```

```
        tnum = 2;
    case 'Position'
        tnum = 3;
end
set(handles.popupmenu3,'Value',tnum);

% Update handles structure
guidata(hObject, handles);


% UIWAIT makes configuracio_kinect wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function
varargout=configuracio_kinect_OutputFcn(hObject,eventdata,handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

varargout{1} = [];



% --- Executes on button press in buttonCancelar.
function buttonCancelar_Callback(hObject, eventdata, handles)

% hObject    handle to buttonCancelar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close('configuracio_kinect');
% --- Executes on button press in buttonAcceptar.
function buttonAcceptar_Callback(hObject, eventdata, handles)
%popMenu = get(handles.popupmenu1,'Value');

angle = str2double(get(handles.editAngle,'String'));
 % Obtenim la paraula del desplegable
strBodyPosture = get(handles.popupmenu1,'String');
% Obtenim el valor selecionat com enter del desplegable
posBodyPosture = get(handles.popupmenu1,'Value');
% Obtenim la paraula del desplegable
strTrackMode = get(handles.popupmenu3,'String');
% Obtenim el valor selecionat com enter del desplegable
posTrackMode = get(handles.popupmenu3,'Value');
assignin('base','angleCamera',angle);
% Determina quina es la postura del popupmenu
assignin('base','bodyPosture',strBodyPosture{posBodyPosture});
%i la passa al workspace
assignin('base','trackingMode',strTrackMode{posTrackMode});


close('configuracio_kinect');
% close(configuracio_kinect);
% hObject    handle to buttonAcceptar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editAngle_Callback(hObject, eventdata, handles)
% hObject    handle to editAngle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editAngle as text
%        str2double(get(hObject,'String')) returns contents of editAngle
        ...as a double


% --- Executes during object creation, after setting all properties.
function editAngle_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editAngle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in pmTrackMode.

% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

9.10.3 Funcions per a calcular els angles

El codi de les funcions que s'han definit per calcular els diferents angles del cos amb les dades de la Kinect seran descrites a continuació.

9.10.3.1 angleColzeEsquerre

```
function [ elAngle ] = angleColzeEsquerre( realCoordSkeleton)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
      shlPoint = realCoordSkeleton(5,:,1); % Ens dona el valor del
punt del hombro
       elPoint = realCoordSkeleton(6,:,1); % Ens dona el valor del
colze esquerre
       wlPoint = realCoordSkeleton(7,:,1); % ens dona el valor del
canell esquerre
       elShlVector = shlPoint - elPoint; % Calcula les components del
vector colze i hhombro esquerre
       elWlVector =   wlPoint-elPoint; % Calcula les components del
vector colze i canell dret
       producteEscalar_elAngle=dot(elShlVector,elWlVector); % producte
escalar dels dos vectors que surten del colze
       angleRadians =
acos(producteEscalar_elAngle/(norm(elShlVector,2)*norm(elWlVector,2)))
; % Calcula l'angle en radians del colze mitjançant la formula
       % del producte escalar
       elAngle = angleRadians*180/pi;
end
```

9.10.3.2 angleColzeDret

```
function [erAngle] = angleColzeDret( realCoordSkeleton )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
       shrPoint = realCoordSkeleton(9,:,1); % Ens dona el valor del
punt del hombro
       erPoint = realCoordSkeleton(10,:,1); % Ens dona el valor del
colze dret
       wrPoint = realCoordSkeleton(11,:,1); % ens dona el valor del
canell dret
       erShrVector = shrPoint - erPoint; % Calcula les components del
vector colze i hhombro dret
       erWrVector =   wrPoint-erPoint; % Calcula les components del
vector colze i canell dret
       producteEscalar_elAngle=dot(erShrVector,erWrVector); % producte
escalar dels dos vectors que surten del colze
```

```
        angleRadians =
acos(producteEscalar_elAngle/(norm(erShrVector,2)*norm(erWrVector,2)))
; % Calcula l'angle en radians del colze mitjançant la formula
        % del producte escalar
        erAngle = angleRadians*180/pi;
end
```

9.10.3.3 angleHombroEsquerre

```
function [ shlAngle ] = angleHombroEsquerre( realCoordSkeleton )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
        shlPoint = realCoordSkeleton(5,:,1); % Ens dona el valor del
punt
        ...del espatlla
        elPoint = realCoordSkeleton(6,:,1); % Ens dona el valor del
colze
        ...esquerre
        constantVector = [0 1 0];
        %constantPoint = constantVector - shlPoint;
        elShlVector = shlPoint - elPoint; % Calcula les components del
        ...vector colze i espatlla esquerre
         %producte escalar dels dos vectors que surten del colze
        producteEscalar_elAngle=dot(elShlVector, constantVector);
        %Calcula l'angle en radians del colze mitjançant la formula
        %del producte escalar
        angleRadians = acos(producteEscalar_elAngle/...
            (norm(elShlVector,2)*norm( constantVector,2)));
        shlAngle = angleRadians*180/pi;
end
```

9.10.3.4 angleHombroDret

```
function [ shrAngle ] = angleHombroDret( realCoordSkeleton)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
        shrPoint = realCoordSkeleton(9,:,1); % Ens dona el valor del
punt del hombro
        erPoint = realCoordSkeleton(10,:,1); % Ens dona el valor del
colze dret
        constantVector = [0 1 0];
        erShrVector = shrPoint - erPoint; % Calcula les components del
vector colze i hhombro esquerre
        producteEscalar_elAngle=dot(erShrVector, constantVector); %
producte escalar dels dos vectors que surten del colze
        angleRadians =
acos(producteEscalar_elAngle/(norm(erShrVector,2)*norm(
constantVector,2))); % Calcula l'angle en radians del colze mitjançant
la formula
        % del producte escalar
        shrAngle = angleRadians*180/pi;
end
```

9.10.3.5 angleRollHombroEsquerre

```
function [ shlYawAngle ] = angleRollHombroEsquerre(realCoordSkeleton)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
        shlPoint = realCoordSkeleton(5,:,1); % Ens dona el valor del
punt del hombro
```

```
        hlPoint = realCoordSkeleton(8,:,1); % ens dona el valor de la
mà esquerre
       % shrPoint = realCoordSkeleton(9,:,1);
        shlHlVector = -(hlPoint-shlPoint); % Calcula les components del
vector hombro esquerre i mà esquerre
        shlShrVector =  [1 0 0];  %shrPoint-shlPoint; % Calcula les
components del vector hombro esquerre i hombro dret
        producteEscalar_elAngle=dot(shlHlVector,shlShrVector); %
producte escalar dels dos vectors que surten del hombro
        angleRadians =
acos(producteEscalar_elAngle/(norm(shlHlVector,2)*...
            norm(shlShrVector,2))); % Calcula l'angle en radians del
colze mitjançant la formula
        % del producte escalar
        shlYawAngle = angleRadians*180/pi;
end
```

9.10.3.6 angleRollHombroDret

```
function [ shrYawAngle ] = angleRollHombroDret(realCoordSkeleton)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
     shrPoint = realCoordSkeleton(9,:,1); % Ens dona el valor del punt
del hombro
        hrPoint = realCoordSkeleton(12,:,1); % ens dona el valor de la
mà esquerre
       % shrPoint = realCoordSkeleton(9,:,1);
        shrHrVector = hrPoint-shrPoint; % Calcula les components del
vector hombro esquerre i mà esquerre
        shrShlVector =  [1 0 0];  %shrPoint-shlPoint; % Calcula les
components del vector hombro esquerre i hombro dret
        producteEscalar_elAngle=dot(shrHrVector,shrShlVector); %
producte escalar dels dos vectors que surten del hombro
        angleRadians =
acos(producteEscalar_elAngle/(norm(shrHrVector,2)*...
            norm(shrShlVector,2))); % Calcula l'angle en radians del
colze mitjançant la formula
        % del producte escalar
        shrYawAngle = angleRadians*180/pi;
end
```

9.11 Programes i scripts creats per al Leap Motion

9.11.1 Proves

```
%Script per mostrar la posició de la mà sobre el pla x i y; i x i z
f = figure;
%Inicialització dels gràfics
subplot(2,1,1);
axis([-300 300 0 500]);
xlabel('x');
ylabel('y');
grid on;

subplot(2,1,2);
axis([-300 300 -400 400]);
xlabel('x');
ylabel('z');
grid on;

%Bucle principal
while ishandle(f)
  metaData =  matleap(1); %Agafar dades
  if ~isempty(metaData.pointables) %Comprobació si detecta dades
      x=metaData.pointables(3).position(1);
      y=metaData.pointables(3).position(2);
      z=metaData.pointables(3).position(3);
      x = round(x);
      y = round(y);
      z = round(z);
      %Actualitzar els gràfics
      subplot(2,1,1);
      plot(x,y,'+r','LineWidth',2),axis([-300 300 0 500]);
      text(x,y,['(' num2str(x) ',' num2str(y) ')']);
      xlabel('x');
      ylabel('y');
      grid on;
      subplot(2,1,2);
      plot(x,z,'+g','LineWidth',2),axis([-300 300 -400 400]);
      text(x,z,['(' num2str(x) ',' num2str(z) ')']);
      xlabel('x');
      ylabel('z');
      grid on;
      drawnow;
  end
  pause(0.001);
end
```

9.11.2 Programa Principal

```
function varargout = programaTrajectoria(varargin)
% PROGRAMATRAJECTORIA MATLAB code for programaTrajectoria.fig
%      PROGRAMATRAJECTORIA, by itself, creates a new PROGRAMATRAJECTORIA
or raises the existing
%      singleton*.
%
%      H = PROGRAMATRAJECTORIA returns the handle to a new
PROGRAMATRAJECTORIA or the handle to
%      the existing singleton*.
%
```

```
%      PROGRAMATRAJECTORIA('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in PROGRAMATRAJECTORIA.M with the given
input arguments.
%
%      PROGRAMATRAJECTORIA('Property','Value',...) creates a new
PROGRAMATRAJECTORIA or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before programaTrajectoria_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to programaTrajectoria_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
 % Edit the above text to modify the response to help
programaTrajectoria

% Last Modified by GUIDE v2.5 11-May-2015 02:18:43

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @programaTrajectoria_OpeningFcn,
...
                   'gui_OutputFcn',  @programaTrajectoria_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before programaTrajectoria is made visible.
function programaTrajectoria_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to programaTrajectoria (see
VARARGIN)


% Choose default command line output for programaTrajectoria
%Llimpiem pantalla
```

```
 instrreset
 clc

handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes programaTrajectoria wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = programaTrajectoria_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in buttonStart.
function buttonStart_Callback(hObject, eventdata, handles)
% hObject    handle to buttonStart (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Definció de variables
global despegat... Variable booleana per a controlar l'estat del drone
       speed...
       yaw pitch... Angles yaw i pitch del dit del mig
       is_Moving... Comproba si s'esta movent
       x z... Variables de posició del dit del mig
       u v w... Components del vector de direcció i sentit del dit del
mig
       ar... Objecte de la classe ARDrone
       j i... Contadors
       time  totalTime... variables per a calcular el temps
       altitud Vx Vy... Variables d'altitud i velocitat als eixos x i y
       rightTilt leftTilt reverseTilt forwardTilt... %Angles per enviar-
li
       timer2 timer3 %Timers per mostrar dades per pantalla

    if strcmp(get(hObject,'String'),'Start')
        %Inicialització de variables
        set(hObject,'String','Stop');
        ar = ARDrone;
        despegat = 0;
        speed = 1;
        is_Moving = 1;
        sumaDits = 0;
        firstTime = 1;
        j = 1;
        i = 1;
        rightTilt=0;
        leftTilt=0;
        reverseTilt=0;
        forwardTilt=0;
```

```
        x(i)=0; z(i)=0; u(i)=0; v(i)=0; w(i)=0;
        Vx(j)=0;        Vy(j)=0;
        timer2=tic;
        timer3=tic;
    elseif strcmp(get(hObject,'String'),'Stop')

        if  ar.NavData ~= 0 %Si s'envien dades
          ar.land(); %Cridar a la funció stop de la clase ARDrone.
        end
        clear ar; %Eliminar l'objecte.
        set(hObject,'String','Start');


           uisave({'totalTime','Vx','Vy','altitud','x','z','v'},...
                  'dadesTrajectoria');

    end
   time = tic;
   %Inicialization axis

     handles.xzPlot=plot(handles.axesHandPosition,x(i),-z(i)...
         ,'+g','LineWidth',20);
        axis(handles.axesHandPosition,[-350 350 -350 350]);
       handles.wvCom=compass(handles.axesPitch,w(i),v(i),'r');
        axis(handles.axesPitch,'manual',[-1 1 -1 1]);
       set(handles.wvCom,'LineWidth',3);
       handles.uwCom=compass(handles.axesYaw,u(i),w(i),'b');
        axis(handles.axesYaw,'manual',[-1 1 -1 1]);
        set(handles.uwCom,'LineWidth',3);

  while strcmp (get(hObject,'String'),'Stop') %Bucle principal
         metaData =  matleap(1); %Retorna un frame
  if ~isempty(metaData.pointables) %Comprobem que el L.P. enviï dades

      for cont=1:5 %Bucle per sumar quants dits hi ha allargats
          sumaDits = sumaDits+metaData.pointables(cont).is_extended;
      end

      x(i)=metaData.pointables(3).position(1);
      z(i)=metaData.pointables(3).position(3);
      %Arredoniment de variables
      x(i) = round(x(i));
      z(i) = round(z(i));

      % Control Yaw and Pitch del dit del mig
      u(i)=metaData.pointables(3).direction(1);
      v(i)=metaData.pointables(3).direction(2);
      w(i)=metaData.pointables(3).direction(3);
      %Arredoniment de variables
      u(i) = round(u(i),2);
      v(i) = round(v(i),2);
      w(i) = round(w(i),2);

      %Mostrar per pantalla dades Leap Motion
      if toc(timer3)>0.033
            set(handles.xzPlot,'XData',x(i),'YData',-z(i));
            set(handles.wvCom,'XData',[0 w(i)],'YData',[0 v(i)]);
            set(handles.uwCom,'XData',[0 u(i)],'YData',[0 w(i)]);
      end
```

```
        if sumaDits == 2 %Si mes de 2 dits allargats
            if firstTime == 1
                firstTime = 0;
                elpasedTime1=tic;

            elseif toc(elpasedTime1) >= 2 %Contador de dos segons
                 if despegat == 0    %Despega si està aterrat
                      ar.takeoff();

set(handles.textDroneEstat,'String','Despegant...',...
                          'foregroundcolor',[1 0.6 0.2]);
                      pause(4); % Espera uns segons a què despegui
                      despegat = 1;
                      set(handles.textDroneEstat,'String','Despegat',...
                          'foregroundcolor',[0 1 0]);
                      firstTime =1;
                 elseif despegat == 1 %Aterra si està despegat
                      ar.land();

set(handles.textDroneEstat,'String','Aterrant...',...
                          'foregroundcolor',[1 0.6 0.2]);
                      pause(2);
                      despegat = 0;
                        set(handles.textDroneEstat,'String','Aterrat',...
                          'foregroundcolor',[1 0 0]);
                      firstTime = 1;
                 end
            end
        elseif firstTime ==0
            firstTime =1;
        end

 %---Realitzar control del drone--------------------
     if despegat == 1 && sumaDits > 2 %Si tots els dits extesos
      %Anar cap a la dreta
      if x(j) > 70
          rightTilt = 0.0146*x(j)-0.9231; % 0.1<=rightTilt<=2
          ...and 70<x<=200
          if rightTilt > 2
              rightTilt = 2;
          end
          ar.moveRight(rightTilt);
          is_Moving = 1;
      %Anar cap a l'esquerra
      elseif x(j) < -70
          leftTilt = -0.0146*x(j)-0.9231; % 0.1<=leftTilt<=2
          ...and -70<x<=-200
          if leftTilt > 2
              leftTilt = 2;
          end
          ar.moveLeft(leftTilt);
          is_Moving = 1;
      end
      %Anar cap endavant
      if z(j) > 45
          reverseTilt =  0.0123*z(j)-0.4516; % 0.1<=reverseTilt<=2 and
45<z<=200
          if reverseTilt > 2
              reverseTilt = 2;
```

```
            end
            ar.moveReverse(reverseTilt);
            is_Moving = 1;
        %Anar cap endarrere
        elseif z(j) < -70
            forwardTilt =  -0.0146*z(j)-0.9231; % -0.1<=reverseTilt<=-2

            ...and -70<z<=-200
             if forwardTilt >2
                forwardTilt = 2;
            end
            ar.moveForward(forwardTilt);
            is_Moving = 1;
        end
        %Rotar cap a la dreta
        if u(j) > 0.4
            yaw = 3.1667*u(j)-1.1667; % 0.2<=yaw<=2 and  0.4<u<1
            ar.rotateRight(yaw);
            is_Moving = 1;
        %Rotar cap a l'esquerra
        elseif u(j) < -0.4
            yaw = -3.1667*u(j)-1.1667;  % 0.2<=yaw<=2 and  -0.4<u<-1
            ar.rotateLeft(yaw);
            is_Moving = 1;
        end
        %Anar amunt
        if v(j) > 0.4
            pitch = 3.1667*v(j)-1.1667; % 0.2<=pitch<=2 and  0.4<v<1
            ar.moveUp(pitch);
        %Anar avall
        elseif v(j) < -0.4
            pitch = -3.1667*v(j)-1.1667; %0.2<=pitch<=2 and  -0.4<v<-1
            ar.moveDown(pitch);
            is_Moving = 1;
        end

      end
      sumaDits =0; %Reinici de la variable que conta els dits

        elseif despegat == 1 && is_Moving == 1% If the programe don't
track
        ...any data, drone will land
        ar.hover();
        is_Moving = 0;
   end
   % Actualitzar dades de navegació

   Vx(j) = ar.X_Velocity;
   Vy(j) = ar.Y_Velocity;
   altitud(j) = ar.Altitude;

   %Refrescar la pantalla amb les dades del Drone
   if toc(timer2) > 0.033

       set(handles.textBateriaValor,'String',...
                    [num2str(ar.Battery_Voltage) '%']);
       set(handles.textPitchValor,'String',...
                    [num2str(round(ar.Pitch)) '°']);
       set(handles.textRollValor,'String',...
                    [num2str(round(ar.Roll)) '°']);
```

```
      set(handles.textYawValor,'String',...
                    [num2str(round(ar.Yaw)) '°']);
      set(handles.textAltitudValor,'String',...
                    [num2str(round(ar.Altitude,3)) ' m']);
      set(handles.textVxValor,'String',...
                    [num2str(round(ar.X_Velocity,4)) ' m/s']);
      set(handles.textVyValor,'String',...
                    [num2str(round(ar.Y_Velocity,4)) ' m/s']);
      timer2 = tic; %Reiniciem timer
  end

   totalTime(j)=toc(time); %Temps total de programa
   j = j+1;
   pause(0.000001); %Pause per evitar bloqueig del programa al bucle
   end

% --- Executes on button press in buttonMostrarTrajectoria.
function buttonMostrarTrajectoria_Callback(hObject, eventdata, handles)
% hObject    handle to buttonMostrarTrajectoria (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(handles.buttonStart,'String'),'Start')
    load('dadesTrajectoria');
    pause(0.5);
    run('mostrarTrajectoria');
end

 % --- Executes during object creation, after setting all properties.
function axesHandPosition_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axesHandPosition (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: place code in OpeningFcn to populate axesHandPosition

 % --- Executes during object creation, after setting all properties.
function axesPitch_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axesPitch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: place code in OpeningFcn to populate axesPitch

 % --- Executes during object creation, after setting all properties.
function axesYaw_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axesYaw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: place code in OpeningFcn to populate axesYaw

% --- Executes during object creation, after setting all properties.
function textDroneEstat_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textDroneEstat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
 % --- Executes during object creation, after setting all properties.
function textBateriaValor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textBateriaValor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

 % --- Executes during object creation, after setting all properties.
function textPitchValor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textPitchValor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

 % --- Executes during object creation, after setting all properties.
function textRollValor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textRollValor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

 % --- Executes during object creation, after setting all properties.
function textYawValor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textYawValor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

 % --- Executes during object creation, after setting all properties.
function textAltitudValor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textAltitudValor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

 % --- Executes during object creation, after setting all properties.
function textVxValor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textVxValor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

 % --- Executes during object creation, after setting all properties.
function textVyValor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textVyValor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

 % --- Executes on button press in buttonTrajectoriaAnimada.
function buttonTrajectoriaAnimada_Callback(hObject, eventdata, handles)
% hObject    handle to buttonTrajectoriaAnimada (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(handles.buttonStart,'String'),'Start')
    clear all;
    load('dadesTrajectoria'); %Carreguem dades al workSpace
    pause(0.5);
    run('trajectoriaAnimada');
```

```
end
```

9.11.2.1 Script "Trajectòria Animada"

```
%Mostrar Trajectoria Animada

global  j Xpos Ypos  Xact Yact Altitudf
%incialitzacio de variables
Xanterior=0;
Yanterior=0;
Vxanterior =0;
Vyanterior=0;
timeAnterior = 0;
longitud=length(Vx);

for i=1:longitud %càlcul de la trajectòria
    Xact(i) = Xanterior+Vxanterior*(totalTime(i)-timeAnterior)...
        +(Vx(i)-Vxanterior)*(totalTime(i)-timeAnterior)/2;
    Yact(i) = Yanterior+Vyanterior*(totalTime(i)-timeAnterior)...
        +(Vy(i)-Vyanterior)*(totalTime(i)-timeAnterior)/2;
    Xanterior = Xact(i);
    Yanterior = Yact(i);
    Vxanterior = Vx(i);
    Vyanterior = Vy(i);
    timeAnterior = totalTime(i);
end

Xact = round(Xact,3);
Yact= round(Yact,3);
altitud = round(altitud,3);

arrayLength=length(Vx); %Calcular la longitud de Vx es la mateixa que
Vy
... totalTime
j = 1;

h = figure;
axis([-inf inf -inf inf -inf inf]);
title('Trajectoria del Drone')
xlabel('x(m)');
ylabel('y(m)');
zlabel('z(m)');
grid on;

cutOffFrequency = 10;
tsampleFrequency = 30;
normalizedFrequency = cutOffFrequency*2/tsampleFrequency;

[b,a]=butter(2,normalizedFrequency);
Xact=filter(b,a,Xact);
Yact=filter(b,a,Yact);

%Mostrem la gràfica
while ishandle(h) && j<= arrayLength %Mentre la figura no estigui
tancada
    ...executarà la trajectoria

        Xpos(j)=Xact(j);
        Ypos(j)=Yact(j);
```

```
        Altitudf(j)=altitud(j);
        plot3(Xpos,Ypos,Altitudf);   %Realitzem un gràfic de la
trajectòria
        text(Xpos(j),Ypos(j),Altitudf(j),...
            ['(' num2str(Xpos(j)) ', ' num2str(Ypos(j)) ', '...
            num2str(Altitudf(j)) ')']);
        title('Trajectòria Drone');
        xlabel('x(m)');
        ylabel('y(m)');
        zlabel('z(m)');
        grid on;
 pause(0.0001);
 j = j+1;

end
```

9.11.2.2 Script "Mostrar Trajectòria"

```
%mostrar trajectòria dades normals i filtrades
global Xf Yf Altituda Xact Yact
%incialització de varaibles
Xanterior=0;
Yanterior=0;
Vxanterior =0;
Vyanterior=0;
timeAnterior = 0;
longitud=length(Vx);
%Càlcul de la trajectòria
for i=1:longitud
    Xact(i) = Xanterior+Vxanterior*(totalTime(i)-timeAnterior)+...
        (Vx(i)-Vxanterior)*(totalTime(i)-timeAnterior)/2;
    Yact(i) = Yanterior+Vyanterior*(totalTime(i)-timeAnterior)...
        +(Vy(i)-Vyanterior)*(totalTime(i)-timeAnterior)/2;
    Xanterior = Xact(i);
    Yanterior = Yact(i);
    Vxanterior = Vx(i);
    Vyanterior = Vy(i);
    timeAnterior = totalTime(i);
end

Xact = round(Xact,3);
Yact= round(Yact,3);
altitud = round(altitud,3);
Xa = Xact;
Ya = Yact;

%creació del filtre i els seus valors
 cutOffFrequency = 1;
 SamplingRate = 30;
 NormalizedFrequency = cutOffFrequency*2/SamplingRate;
 [b,a] = butter(4,NormalizedFrequency); %Creem un butterwoth per
filtrar les dades

   Xf=filter(b,a,Xa); %Filtrar coordenada x
   Yf=filter(b,a,Ya); %FIltrar coordenada y
   Altituda=filter(b,a,altitud); %Filtrar coodenada z
   Xf=round(Xf,3);
   Yf=round(Yf,3);
   Altituda=round(Altituda,3);
```

```
    DadesSenseFiltrar = figure;

%Plot que mostra les dades agafades

 plot3(Xact,Yact,altitud);   %Realitzem un gràfic de la trajectòria
     title('Trajectòria Drone');
        xlabel('x(m)');
        ylabel('y(m)');
        zlabel('z(m)');
        grid on;

%Gràfic que contindrà les dades filtrades
DadesFiltrades=figure;

        plot3(Xf,Yf,Altituda),axis([-inf inf -inf 0 inf]);

        title('Trajectòria Filtrada');
        xlabel('x(m)');
        ylabel('y(m)');
        zlabel('z(m)');
        grid on;
```

9.11.3 Programa Arduino

```
%Programa que compta el número de dits allargats d'una mà
run = 1; %Variable booleana per fer correr el bucle principal
arduinoUno = arduino('COM4'); %Connectem amb Arduino
extendedFingers = 0; %Variable que compta els dits allargats
%Inicialitzem els pins de l'Arduino
if ~isempty(arduinoUno)
    arduinoUno.pinMode(3,'output');
    arduinoUno.pinMode(4,'output');
    arduinoUno.pinMode(5,'output');
    arduinoUno.pinMode(6,'output');
    arduinoUno.pinMode(7,'output');
end
%Bucle Principal
while run
  metaData =  matleap_frame; %Agafem Dades
  if ~isempty(metaData.pointables) && ~isempty(arduinoUno)
     %Comptador de dits
    for i=1:length(metaData.pointables)

extendedFingers=extendedFingers+metaData.pointables(i).is_extended;
    end
  end
  % Switch que analitza els 6 possibles estats
  switch extendedFingers
      case 0 % cap dit allargat
           arduinoUno.digitalWrite(3,0);
            arduinoUno.digitalWrite(4,0);
              arduinoUno.digitalWrite(5,0);
                arduinoUno.digitalWrite(6,0);
                  arduinoUno.digitalWrite(7,0);
      case 1 %1 dit allargat
          arduinoUno.digitalWrite(3,1);
            arduinoUno.digitalWrite(4,0);
              arduinoUno.digitalWrite(5,0);
                arduinoUno.digitalWrite(6,0);
```

```
                    arduinoUno.digitalWrite(7,0);

        case 2 %2 dits allargats
            arduinoUno.digitalWrite(3,1);
              arduinoUno.digitalWrite(4,1);
                arduinoUno.digitalWrite(5,0);
                  arduinoUno.digitalWrite(6,0);
                    arduinoUno.digitalWrite(7,0);

        case 3 %3 dits allargats
               arduinoUno.digitalWrite(3,1);
              arduinoUno.digitalWrite(4,1);
                arduinoUno.digitalWrite(5,1);
                  arduinoUno.digitalWrite(6,0);
                    arduinoUno.digitalWrite(7,0);

        case 4 %4 dits allargats
            arduinoUno.digitalWrite(3,1);
              arduinoUno.digitalWrite(4,1);
                arduinoUno.digitalWrite(5,1);
                  arduinoUno.digitalWrite(6,1);
                    arduinoUno.digitalWrite(7,0);
        case 5 %5 dits allargats
             arduinoUno.digitalWrite(3,1);
              arduinoUno.digitalWrite(4,1);
                arduinoUno.digitalWrite(5,1);
                  arduinoUno.digitalWrite(6,1);
                    arduinoUno.digitalWrite(7,1);
        otherwise %Més de cinc dits allargats
            run = 0; %Para el programa
    end
     extendedFingers = 0;

end
clear all;
```

9.12 Enllaços a Youtube

Per tal d'il·lustrar al lector els resultats obtinguts, s'han gravat vídeos i s'han penjat a la pàgina web de Youtube. Tots els vídeos han estat creats i editats per l'autor del treball.

9.12.1 Enllaç del vídeo de la Kinect

Control AR Drone 2.0
https://www.youtube.com/watch?v=leJf4ZJpns0

9.12.2 Enllaços dels vídeos del Leap Motion

Control AR Drone 2.0
https://www.youtube.com/watch?v=XoXetec9jpM

Control Arduino UNO
https://www.youtube.com/watch?v=27Je6N-fADU

9.13 Altres

9.13.1 Classe que mostra les funcions bàsiques del AR Drone

Per establir la connexió entre el Matlab i aquest s'utilitzarà el següent arxiu obtingut del següent enllaç Web:

```
classdef ARDrone < handle
    % ARDrone Class
    % Controls AR Parrot Drone
    properties
        seq = 3;
        ARc;
        ARn;
        % NavData Properties
        Control_State = [];
        Battery_Voltage = 0;
        Pitch = 0;
        Roll = 0;
        Yaw = 0;
        Altitude = 0;
        X_Velocity = 0;
        Y_Velocity = 0;
        NavData = 0;
    end

    methods
        function [obj] = ARDrone()
            instrreset
            clc
           global fh;
           %Create Control connection
           obj.ARc = udp('192.168.1.1', 5556, 'LocalPort', 5556);
           fopen(obj.ARc);
           fh = 0;
           %Create NavData connection
           obj.ARn = udp('192.168.1.1', 5554, 'LocalPort', 5554, ...
            'ByteOrder', 'littleEndian', ...
            'InputBufferSize', 500, ...
            'BytesAvailableFcn', {@navPacketRxCallback,obj}, ...
            'DatagramTerminateMode', 'on', ...
            'BytesAvailableFcnMode', 'byte', ...
            'BytesAvailableFcnCount', 24);
            fopen(obj.ARn);

            % Byte to NavData port for initialization
            fwrite(obj.ARn, 1);

            % NavData command to command port
           AR_NAV_CONFIG = ...
sprintf('AT*CONFIG=2,\"general:navdata_demo\",\"TRUE\"\r');
            fprintf(obj.ARc, AR_NAV_CONFIG);

            % Flicker LEDs
            obj.command('LED', '1,5,5,5,5')
          function navPacketRxCallback(obj, event, drone)
            if (get(obj, 'BytesAvailable') ~= 0)
                data = fread(obj);
```

```
                data = uint8(data);
                if ~isempty(data);
                    % Drone Control State Information
                        drone_state = [data(5) data(6) data(7) data(8)];
                        drone_state = typecast(drone_state, 'uint32');
                        for i = 1:32
                           drone.Control_State(i) = bitget(drone_state,
i);
                        end
                    % Battery Voltage Percentage
                      vBattery = [data(25) data(26) data(27) data(28)];
                      drone.Battery_Voltage = typecast(vBattery,
'uint32');
                    % Theta (Pitch) Values
                        tPitch = [data(29) data(30) data(31) data(32)];
                        drone.Pitch = typecast(tPitch, 'single') / 1000;

                    % Phi (Roll) Values
                        tRoll = [data(33) data(34) data(35) data(36)];
                        drone.Roll = typecast(tRoll, 'single') / 1000;

                    % Psi (Yaw) Values
                        tYaw = [data(37) data(38) data(39) data(40)];
                        drone.Yaw = typecast(tYaw, 'single') / 1000;

                    % Altitude
                        tAltitude = [data(41) data(42) data(43)
data(44)];
                        drone.Altitude = ...
                            single(typecast(tAltitude, 'int32'))/ 1000;

                    % X Velocity
                        Vx = [data(45) data(46) data(47) data(48)];
                        drone.X_Velocity = typecast(Vx, 'single') /
1000;

                    % Y Velocity
                        Vy = [data(49) data(50) data(51) data(52)];
                        drone.Y_Velocity = typecast(Vy, 'single') /
1000;

                    % Entire NavData Packet
                        drone.NavData = data;
                end

                % Reset Drone Watchdog Bit
                AR_WDG = strcat('AT*COMWDG=',num2str(drone.seq),',');
                fprintf(drone.ARc, AR_WDG);
                drone.seq = drone.seq + 1;

        %         if fh ~= 0
        %             if strcmp(get(fh, 'Selected'), 'off')
        %                 set(fh, 'Selected', 'on')
        %             end
        %         end
        %
            end
          end
        end
        function command(obj, command_type, code)
```

```
            AR_CMD = sprintf( 'AT*%s=%i,%s\r', command_type, obj.seq,
code);
            fprintf(obj.ARc, AR_CMD);
            obj.seq = obj.seq + 1;
        end
        function takeoff(obj)
            obj.command('FTRIM','')
            obj.command('CONFIG','\"control:altitude_max\", \"20000\"')
            obj.command('REF','290718208')
        end
        function hover(obj)
            obj.drive([0,0,0,0])
        end
        function land(obj)
          obj.command('REF','290717696')
        end
        function emergency(obj)
            obj.command('REF','290717952')
            obj.command('REF','290717696')
        end
        function drive(obj, speed)
            % Controls the robot movement directly using motor speeds
            fvel = typecast(single(speed(1)/2), 'int32');
            lvel = typecast(single(speed(2)/2), 'int32');
            uvel = typecast(single(speed(3)/2), 'int32');
            rvel = typecast(single(speed(4)/2), 'int32');
            fvel_str = strcat(num2str(fvel),',');
            lvel_str = strcat(num2str(lvel),',');
            uvel_str = strcat(num2str(uvel),',');
            rvel_str = num2str(rvel);
            obj.command('PCMD',strcat('1,',  lvel_str,  fvel_str,
uvel_str,  rvel_str))
        end
        function moveUp(obj, speed)
        obj.drive([0,0,speed,0])
        end
        function moveDown(obj, speed)
        obj.drive([0,0,-speed,0])
        end
        function moveLeft(obj, speed)
        % Moves the robot left at a specific speed
        obj.drive([0,-speed,0,0])
        end
        function moveRight(obj, speed)
        % Moves the robot right at a specific speed
        obj.drive([0,speed,0,0])
        end
        function moveForward(obj, speed)
        % Moves the robot forward at a specific speed
        obj.drive([-speed,0,0,0])
        end
        function moveReverse(obj, speed)
        % Moves the robot forward at a specific speed
        obj.drive([speed,0,0,0])
        end
        function rotateLeft(obj, omega)
        % Rotates robot left
            obj.drive([0,0,0, -omega])
        end
        function rotateRight(obj, omega)
            obj.drive([0,0,0, omega])
```

```
        end
        function stop(obj)
        % Stop all Robot motion and close communications.
            obj.land
            fclose(obj.ARn);
            fclose(obj.ARc);
        end
        function control(obj, varargin)
            clc
            disp('Controls');
            disp('arrow keys = up,down,rotate_left,rotate_right');
            disp( 'w,a,s,d = move:forward,backward,left,right');
            disp('enter,spacebar,shift = takeoff,land,hover');
            disp('q = quit program');
        % Manually control robot using W/S (forward/reverse), A/D
        % (strafe left/right), I/K (vertical up/down), J/L (turn
left/right)
        % E (emergency stop), and Q (quit) keys.
            global fh;
            if nargin == 2
                vel = varargin{1};
            else
                vel = .5;
            end
            left = 0;
            bottom = 0;
            width = 100;
            height = 150;
            fh = figure(...
                'name','ARDrone Controller', ...
                'keypressfcn',@keyPress, ...
                'windowstyle','modal',...
                'numbertitle','off', ...
                'Position',[left bottom width*3 height],...
                'userdata','timeout',...
                'Color','white') ;
            t =
timer('TimerFcn',{@display_Nav,obj},'ExecutionMode','fixedRate');
            start(t)
            function keyPress(~, event)
                    %obj.display_Nav();
                    if (strcmp(event.Character, 'q'))
                        obj.command('REF','290717952')
                        stop(t)
                        delete(t)
                        delete(fh)
                        fh = 0;
                    end
                    cmd = event.Key;
                    switch (cmd)
                        case 'return' % Take off
                            obj.takeoff()
                        case 'space' % Land
                            obj.land()
                        case 'e' % Emergency
                            obj.emergency()
                        case 'a' % Rotate Left
                            obj.rotateLeft(vel)
                        case 'd' % Rotate Right
                            obj.rotateRight(vel)
                        case 'uparrow' % Up
```

```
                            obj.moveUp(vel)
                        case 'downarrow' % Down
                            obj.moveDown(vel)
                        case 'shift' % Hover
                            obj.hover()
                        case 'leftarrow' % Left
                            obj.moveLeft(vel)
                        case 'rightarrow' % Right
                            obj.moveRight(vel)
                        case 'w' % Forward
                            obj.moveForward(vel)
                        case 's' % Backward
                            obj.moveReverse(vel)
                        case 'q'
                            disp('Quitting...')
                        otherwise
                            disp('Unknown Command')
                    end
            end
        function display_Nav(value,event,obj)
            left = 0;
            bottom = 0;
            width = 100;
            height = 150;
            labels = uicontrol('Style','text','Position',[left bottom
width height],'HorizontalAlignment', 'left', 'FontSize',
14,'BackgroundColor','white');
            str1 = 'Battery';
            str2 = 'Pitch';
            str3 = 'Yaw';
            str4 = 'Roll';
            str5 = 'Altitude';
            str6 = 'X_Velocity';
            str7 = 'Y_Velocity';
            set(labels,'String',{str1,str2,str3,str4,str5,str6,str7});

            values = uicontrol('Style','text','Position',[left+width
bottom width height],'HorizontalAlignment', 'right', 'FontSize',
14,'BackgroundColor','white');
            str1 = num2str(obj.Battery_Voltage);
            str2 = num2str(obj.Pitch);
            str3 = num2str(obj.Yaw);
            str4 = num2str(obj.Roll);
            str5 = num2str(obj.Altitude);
            str6 = num2str(obj.X_Velocity);
            str7 = num2str(obj.Y_Velocity);
            set(values,'String',{str1,str2,str3,str4,str5,str6,str7});

            units =
uicontrol('Style','text','Position',[left+width+width+10 bottom width
height],'HorizontalAlignment', 'left', 'FontSize',
14,'BackgroundColor','white');
            str1 = num2str('volts');
            str2 = num2str('degrees');
            str3 = num2str('degrees');
            str4 = num2str('degrees');
            str5 = num2str('meters');
            str6 = num2str('meters/sec');
            str7 = num2str('meters/sec');
            set(units,'String',{str1,str2,str3,str4,str5,str6,str7});
    end
```

```
            end
        end
    end
```

9.13.2 Arxius auxiliars del Leap Motion

9.13.2.1 "matleap.h"

```
/// @file matleap.h
/// @brief leap motion controller interface
/// @author Jeff Perry <jeffsp@gmail.com>
/// @version 1.0
/// @date 2013-09-12

#ifndef MATLEAP_H
#define MATLEAP_H

#define MAJOR_REVISION 0
#define MINOR_REVISION 6

#include "Leap.h"
#include "mex.h"

namespace matleap
{

/// @brief a leap frame
struct frame
{
    int64_t id;
    int64_t timestamp;
    Leap::PointableList pointables; // Give position from fingers and
tools
    Leap::HandList hands; //Get hand frames
};

/// @brief leap frame grabber interface
class frame_grabber
{
    private:
    bool debug;
    Leap::Controller; // Get frames and data
    frame current_frame;
    public:
    /// @brief constructor
    frame_grabber ()
        : debug (false)
    {
        // receive frames even when you don't have focus
        controller.setPolicyFlags
(Leap::Controller::POLICY_BACKGROUND_FRAMES);
    }
    /// @brief destructor
    ~frame_grabber ()
    {
        if (debug)
            mexPrintf ("Closing matleap frame grabber\n");
    }
    /// @brief debug member access
    ///
    /// @param flag turn it on/off
    void set_debug (bool flag)
    {
        if (flag == debug)
            return;
```

```
        if (flag)
            mexPrintf ("Setting debug on\n");
        debug = flag;
    }
    /// @brief get a frame from the controller
    ///
    /// @return the frame
    const frame &get_frame ()
    {
        const Leap::Frame &f = controller.frame (); //f = frames
        current_frame.id = f.id ();
        if (debug)
            mexPrintf ("Got frame with id %d\n", current_frame.id);
        current_frame.timestamp = f.timestamp ();
        current_frame.pointables = f.pointables ();
        return current_frame;
    }
};

} // namespace matleap

#endif
```

9.13.2.2 "matleap.cpp"

```
/// @file matleap.cpp
/// @brief leap motion controller interface
/// @author Jeff Perry <jeffsp@gmail.com>
/// @version 1.0
/// @date 2013-09-12

#include "matleap.h"

// Under Windows, a Leap::Controller must be allocated after the MEX
// startup code has completed.  Also, a Leap::Controller must be
// deleted in the function specified by mexAtExit after all global
// destructors are called.  If the Leap::Controller is not allocated
// and freed in this way, the MEX function will crash and cause MATLAB
// to hang or close abruptly.  Linux and OS/X don't have these
// constraints, and you can just create a global Leap::Controller
// instance.

// Global instance pointer
matleap::frame_grabber *fg = 0;

// Exit function
void matleap_exit ()
{
    delete fg;
    fg = 0;
}

/// @brief process interface arguments
///
/// @param nlhs matlab mex output interface
/// @param plhs[] matlab mex output interface
/// @param nrhs matlab mex input interface
/// @param prhs[] matlab mex input interface
///
/// @return command number
```

```
int get_command (int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
    int command;
    // check inputs
    switch (nrhs)
    {
        case 1:
        command = *mxGetPr (prhs[0]);
        break;
        case 0:
        mexErrMsgTxt ("Not enough input arguments");
        default:
        mexErrMsgTxt ("Too many input arguments");
    }
    // check that inputs agree with command
    switch (command)
    {
        case -1:
        {
            // set debug command requires 0 outputs
            if (nlhs != 0)
                mexErrMsgTxt ("Wrong number of outputs specified");
        }
        break;
        case 0:
        {
            // get version command requires 1 outputs
            if (nlhs != 0 && nlhs != 1)
                mexErrMsgTxt ("Wrong number of outputs specified");
        }
        break;
        case 1:
        {
            // frame grab command only requires one input
            if (nrhs > 1)
                mexErrMsgTxt ("Too many inputs specified");
            // frame grab command requires exactly one output
            if (nlhs != 0 && nlhs != 1)
                mexErrMsgTxt ("Wrong number of outputs specified");
        }
        break;
        default:
        mexErrMsgTxt ("An unknown command was specified");
    }
    return command;
}

/// @brief helper function
///
/// @param v values to fill array with
///
/// @return created and filled array
mxArray *create_and_fill (const Leap::Vector &v)
{
    mxArray *p = mxCreateNumericMatrix (1, 3, mxDOUBLE_CLASS, mxREAL);
    *(mxGetPr (p) + 0) = v.x;
    *(mxGetPr (p) + 1) = v.y;
    *(mxGetPr (p) + 2) = v.z;
    return p;
}
```

```
/// @brief get a frame from the leap controller
///
/// @param nlhs matlab mex output interface
/// @param plhs[] matlab mex output interface
void get_frame (int nlhs, mxArray *plhs[])
{
    // get the frame
    const matleap::frame &f = fg->get_frame ();

    // create matlab frame struct
    const char *frame_field_names[] =
    {
        "id",
        "timestamp",
        "pointables"
    };
    int frame_fields = sizeof (frame_field_names) / sizeof
(*frame_field_names);
    plhs[0] = mxCreateStructMatrix (1, 1, frame_fields,
frame_field_names);
    // fill the frame struct
    mxSetFieldByNumber (plhs[0], 0, 0, mxCreateDoubleScalar (f.id));
    mxSetFieldByNumber (plhs[0], 0, 1, mxCreateDoubleScalar
(f.timestamp));

    // create the pointables structs

    if (f.pointables.count () > 0)
    {
        const char *pointable_field_names[] =
        {
            "id",
            "position",
            "velocity",
            "direction",
            "is_extended",
            "is_finger",
            "is_tool",
            "is_valid",
            "length",
            "width",
            "touch_distance",
            "time_visible"
        };
        int pointable_fields = sizeof (pointable_field_names) / sizeof
(*pointable_field_names);
        mxArray *p = mxCreateStructMatrix (1, f.pointables.count (),
pointable_fields, pointable_field_names);
        mxSetFieldByNumber (plhs[0], 0, 2, p);
        // fill the pointables structs



        for (size_t i = 0; i < f.pointables.count (); ++i)
        {
            // set the id
            mxSetFieldByNumber (p, i, 0, mxCreateDoubleScalar
(f.pointables[i].id ()));
            // create and fill arrays for vectors
```

```
            mxArray *pos = create_and_fill
(f.pointables[i].tipPosition ());
            mxArray *vel = create_and_fill
(f.pointables[i].tipVelocity ());
            mxArray *dir = create_and_fill (f.pointables[i].direction
());

            // set them in the struct
            mxSetFieldByNumber (p, i, 1, pos);
            mxSetFieldByNumber (p, i, 2, vel);
            mxSetFieldByNumber (p, i, 3, dir);
            mxSetFieldByNumber (p, i, 4, mxCreateDoubleScalar
(f.pointables[i].isExtended ()));
            mxSetFieldByNumber (p, i, 5, mxCreateDoubleScalar
(f.pointables[i].isFinger ()));
            mxSetFieldByNumber (p, i, 6, mxCreateDoubleScalar
(f.pointables[i].isTool ()));
            mxSetFieldByNumber (p, i, 7, mxCreateDoubleScalar
(f.pointables[i].isValid ()));
            mxSetFieldByNumber (p, i, 8, mxCreateDoubleScalar
(f.pointables[i].length ()));
            mxSetFieldByNumber (p, i, 9, mxCreateDoubleScalar
(f.pointables[i].width ()));
            mxSetFieldByNumber (p, i, 10, mxCreateDoubleScalar
(f.pointables[i].touchDistance ()));
            mxSetFieldByNumber (p, i, 11, mxCreateDoubleScalar
(f.pointables[i].timeVisible ()));


        }

    }

}

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
    if (!fg)
    {
        fg = new matleap::frame_grabber;
        if (fg == 0)
            mexErrMsgTxt ("Cannot allocate a frame grabber");
        mexAtExit (matleap_exit);
    }
    switch (get_command (nlhs, plhs, nrhs, prhs))
    {
        // turn on debug
        case -1:
        fg->set_debug (true);
        return;
        // get version
        case 0:
        plhs[0] = mxCreateNumericMatrix (1, 2, mxDOUBLE_CLASS,
mxREAL);
        *(mxGetPr (plhs[0]) + 0) = MAJOR_REVISION;
        *(mxGetPr (plhs[0]) + 1) = MINOR_REVISION;
        return;
        // get frame
        case 1:
```

```
            get_frame (nlhs, plhs);
            return;
            default:
            // this is a logic error
            mexErrMsgTxt ("unknown error: please contact developer");
        }
    }
```