

TREBALL FINAL DE MÀSTER  
**Màster en Robòtica**

**HUMAN-AWARE ROBOT BEHAVIOR  
CONTROLLER**

CONTROLADOR DEL COMPORTAMENT D'UN ROBOT TENINT EN COMPTE  
ALS HUMANS

**Autora:**  
JÚLIA MARSAL PERENDREU

**Tutor:**  
NÉSTOR GARCIA HIDALGO

GENER 2021

Santa Coloma de Gramenet, 10 de gener de 2021

## RESUM TREBALL FINAL DE MÀSTER

### MÀSTER EN ROBÒTICA

**Títol:** Controlador del comportament d'un robot tenint en compte als humans

**Paraules clau:** xarxa neuronal, detecció gestos humans, detecció objectes, arbres de comportament

**Autor:** Júlia Marsal Perendreu

**Tutor:** Néstor Garcia Hidalgo (Eurecat)

**Data:** Santa Coloma de Gramenet, 10 de gener del 2021

El present projecte és el resultat del desenvolupament i implementació d'un algoritme de detecció i interacció amb l'humà utilitzant un nou programari base.

El treball es basa en el desenvolupament d'un mòdul cognitiu que permeti el seguiment i la detecció dels gestos d'un humà per tal d'adaptar el comportament del robot a l'acció que estigui desenvolupant l'humà durant el procés de realització d'un cafè. Mitjançant una càmera, una llibreria de detecció de persones i una llibreria de detecció d'objectes el robot entendrà si pot continuar amb la seva tasca, quines tasques pot continuar fent en paral·lel o si es requereix la seva col·laboració per tal de dur a terme una tasca col·laborativa amb l'humà.

Per realitzar entrenaments i proves i per tal d'evitar un deteriorament de l'equip, durant el desenvolupament de l'aplicació es treballa tant en un entorn *local* com *virtual*. I, després, amb l'aplicació acabada, s'efectuen les proves en local.

El present projecte inclou dues parts. En primer lloc, la primera part I conté una breu descripció de les eines de *software* i *hardware* que s'utilitzen en el desenvolupament de l'aplicació així com la filosofia que segueixen i finalment, es fa una breu descripció de l'aplicació a desenvolupar així com la quina arquitectura la conformen. En segon lloc, la segona part [II] inclou la realització i la implementació de l'aplicació.

El desenvolupament d'aquest projecte s'ha fet de forma modular, de tal manera que es pugui anar expandint i optimitzant l'aplicació realitzada de forma que el procés cada cop sigui més robust i tingui en compte una casuística més completa.

Santa Coloma de Gramenet, 10 January 2021

## ABSTRACT

### MÀSTER IN ROBOTICS

**Títol:** Human-aware robot behavior controller

**Paraules clau:** neural networks, human behavior, object detection, behavior tree

**Autor:** Júlia Marsal Perendreu

**Tutor:** Néstor Garcia Hidalgo (Eurecat)

**Data:** Santa Coloma de Gramenet, 10 de gener del 2021

The present project is the result of the development and implementation of an algorithm for detection and interaction with humans using a new software base.

The work is based on the development of a cognitive module that allows the monitoring and detection of the gestures of a human in order to adapt the behavior of the robot to the action that the human is developing during the process of making a coffee. Through a camera, a people detection library and an object detection library the robot will understand if it can continue its task, what tasks can be continued in parallel or if its collaboration is required in order to carry out a collaborative task with the human.

In order to carry out trainings and tests and in order to avoid a deterioration of the equipment, during the development of the application both in a *local* and *virtual* environment are worked on. And then, with the application finished, local testing is performed.

This project includes two parts. First, the first part I contains a brief description of the *software* and *hardware* tools used in application development as well as the philosophy they follow and finally, a brief description of the application to be developed as well as what architecture make it up. Second, the second part [II] includes the realization and implementation of the application.

The development of this project has been done in a modular way, so that the application can be expanded and optimized so that the process is increasingly robust and takes into account a more complete case series.

# Índex

<b>I PART I</b>	<b>10</b>
<b>1 PREFACI</b>	<b>11</b>
1.1 Origen del projecte . . . . .	11
1.2 Motivació . . . . .	11
1.3 Requisits previs . . . . .	11
<b>2 INTRODUCCIÓ</b>	<b>12</b>
2.1 OBJECTIUS DEL PROJECTE . . . . .	12
2.2 ABAST DEL PROJECTE . . . . .	12
<b>3 CONEIXEMENTS PREVIS</b>	<b>14</b>
3.1 ROS . . . . .	14
3.1.1 Nodes . . . . .	14
3.1.2 Packages (Paquets) . . . . .	15
3.1.3 Tòpics . . . . .	15
3.1.4 Accions . . . . .	15
3.1.5 ROS Master . . . . .	15
3.1.6 Namespaces . . . . .	15
3.2 Behavior Tree (Arbre de Comportament) . . . . .	16
3.2.1 Introducció . . . . .	16
3.2.2 Accions i Condicions . . . . .	17
3.2.3 Tick . . . . .	17
3.2.4 Nodes . . . . .	17
3.2.5 Groot . . . . .	18
3.3 Càmera USB . . . . .	19
3.4 YOLO - You Only Look Once . . . . .	19
3.4.1 Descripció . . . . .	19
3.4.2 Darknet . . . . .	20
3.4.3 Google Colab . . . . .	21

3.4.4	Darknet ROS . . . . .	21
3.5	<i>OpenPose</i> . . . . .	22
3.5.1	Descripció . . . . .	22
3.5.2	OpenPose ROS . . . . .	23
3.6	Arquitectura de ROS . . . . .	24
<b>II</b>	<b>PART II</b>	<b>25</b>
<b>4</b>	<b>IDENTIFICACIÓ I CREACIÓ DE L'ENTORN DE TREBALL:</b>	<b>26</b>
4.1	Aplicació a desenvolupar: . . . . .	26
4.1.1	Introducció: . . . . .	26
4.1.2	Arquitectura: . . . . .	27
4.2	Clustering i detecció: . . . . .	28
4.2.1	Entrenament amb google colab . . . . .	28
4.2.2	Integració amb ROS . . . . .	33
4.2.3	Resultat . . . . .	33
<b>5</b>	<b>IDENTIFICACIÓ I CREACIÓ DEL COMPORTAMENT HUMÀ:</b>	<b>34</b>
5.1	Accions i condicions a detectar: . . . . .	34
5.2	Setup de l'algoritme OpenPose . . . . .	34
5.3	Creació del paquet coffee_machine . . . . .	36
5.3.1	Integració d'OpenPose . . . . .	36
5.3.2	Integració del YOLO . . . . .	36
5.3.3	Algoritme per la detecció de la subjecció d'objectes . . . . .	37
5.3.4	Funcions per mostrar el resultat . . . . .	38
5.3.5	Resultat . . . . .	38
<b>6</b>	<b>DESENVOLUPAMENT DEL CONTROLADOR</b>	<b>39</b>
6.1	Creació de l'arbre de comportament . . . . .	39
6.1.1	Accions . . . . .	39
6.1.2	Condicions . . . . .	40
6.1.3	Branques . . . . .	40
6.2	Integració al paquet coffee_machine . . . . .	44
6.3	Creació del tòpic /coffee_machine/Feedback . . . . .	45
<b>7</b>	<b>VALIDACIÓ DEL CONTROLADOR I DELS EXPERIMENTS</b>	<b>49</b>
7.0.1	Primer Experiment: . . . . .	49
7.1	Segon Experiment: . . . . .	54
7.2	Tercer Experiment: . . . . .	57

<b>8 Anàlisi econòmic</b>	<b>59</b>
8.1 Costos . . . . .	59
<b>9 Impacte mediambiental</b>	<b>60</b>
9.1 Desenvolupament: . . . . .	60
9.2 Vida útil . . . . .	60
9.3 Final de la vida útil . . . . .	60
<b>10 Conclusions</b>	<b>61</b>
10.1 Treball Futur . . . . .	62
<b>11 AGRAÏMENTS</b>	<b>63</b>
<b>Annex</b>	<b>64</b>
<b>A Missatges ROS</b>	<b>65</b>
A.1 darknet_ros . . . . .	65
A.2 openpose_ros . . . . .	69
A.3 coffee_machine/State . . . . .	70
<b>B Calibració càmera</b>	<b>71</b>

# Índex de figures

3.1	Exemple bàsic de ROS . . . . .	14
3.2	Exemple bàsic d'un arbre de comportament . . . . .	16
3.3	Example bàsic d'aplicació del Groot . . . . .	18
3.4	Comparació amb altres detectors . . . . .	19
3.5	Resultat d'una imatge classificada amb Yolo . . . . .	20
3.6	Resultat de la detecció d'una xarxa preentrenada . . . . .	20
3.7	Arquitectura del node darknet_ros . . . . .	21
3.8	Resultat de la detecció de l'algoritme openpose . . . . .	23
3.9	Arquitectura del node openpose_ros . . . . .	23
3.10	Arquitectura del ROS a desenvolupar . . . . .	24
4.1	Escenari de treball . . . . .	26
4.2	Diagrama de flux . . . . .	27
4.3	<i>Open Images Dataset d'una tassa de cafè</i> . . . . .	29
4.4	Creació del dataset per a la classe sucre . . . . .	30
4.5	Document etiqueta de la imatge 4.6 . . . . .	30
4.6	Interfície d'usuari <i>LabelImg</i> . . . . .	31
4.7	Estructura del directori <i>darknet</i> propi . . . . .	32
4.8	Resultat de la detecció utilitzant el paquet <i>darknet_ros</i> . . . . .	33
5.1	Resultat de l'acció <i>Extreure el dipòsit d'aigua de la cafetera</i> . . . . .	38
6.1	Distribució dels subprocessos amb l'eina <i>Groot</i> . . . . .	39
6.2	Representació branca <i>OpenCoffeeMachine</i> . . . . .	42
6.4	Representació branca <i>PutCoffeeCup</i> . . . . .	43
6.6	Tòpic /coffe_machine/Feedback . . . . .	46
6.3	Representació branca <i>AutoClean</i> . . . . .	47
6.5	Representació branca <i>CoffeeType</i> . . . . .	48
7.1	Resultat condició <i>IsCleanCupReady</i> . . . . .	50

7.2	Resultat condició <i>IsWaterTankRemoved</i>	51
7.3	Resultat condició <i>IsCoffeCupReady</i>	51
7.4	Resultat condició <i>HasHumanAddedMilk</i>	52
7.5	Resultat condició <i>HasHumanAddedMilk</i>	53
7.6	Resultat condició <i>IsCleanCupReady</i>	54
7.7	Resultat condició <i>IsWaterTankRemoved</i>	55
7.8	Resultat condició <i>IsMarroTankRemoved</i>	55
7.9	Resultat condició <i>IsCoffeCupReady</i>	56
7.10	Resultat condició <i>HasHumanAddedMilk</i>	57
7.11	Resultat condició <i>HasHumanAddedMilk</i>	58
1	Procés de calibració d'una càmera	71

# Índex de taules

4.1	<i>Relació classe amb el nom a OID</i>	29
4.2	<i>Relació classe amb l'ID assignada</i>	31
5.1	<i>Relació d'accions i condicions amb la forma de detecció</i>	35
5.2	<i>Relació d'accions i condicions amb les classes a detectar</i>	37
6.1	Accions que formen part de l'arbre de comportament	40
6.2	Condicions que formen part de l'arbre de comportament	41
6.3	Subprocessos que formen part de l'arbre de comportament	41
8.1	Costos del Material	59
8.2	Costos d'Enginyeria	59

## Part I

# PART I

# Capítol 1

## PREFACI

### 1.1 Origen del projecte

El fet que l'autora efectués l'any 2016 el projecte final del *Grau en Tecnologies Industrielles*[43] a l'*Institut de Robòtica Industrial*[14] amb un robot humanoide *Bioloid*[34] programat en el llençatge de programació C i basat en el framework ROS[35] va despertar l'interès de l'autora cap a l'aprenentatge de nous llenguatges i estructures robòtiques que permetin fer algoritmes més complexos i efectius. [26].

Seguidament, l'autora treballà al *IOC*[44], on participà en la integració de *ROS*[35] amb el robot *YuMi*[1] d'ABB.[27]

Això va despertar un gran interès de l'autora cap a la realització de nous projectes els quals permetessin realitzar aplicacions més complexes i efectives.

### 1.2 Motivació

Ja des de batxillerat, l'autora ha tingut molta inquietud sobre aquest món tan gran i meravellós de la robòtica. Ha anat duent a terme diferents projectes durant la seva carrera professional, com ara robots simples seguidors de línies, una plataforma Stewart, tallers de robòtica educativa, el projecte final de grau amb el robot DarwinOP[26] i la integració de ROS amb el robot yumi d'ABB [27], explicades en l'apartat anterior.

Aquestes aplicacions robotitzades manquen de la dimensió de percepció i no contenen cap intel·ligència pròpia. És aquest fet el qual ha dotat de màxima motivació a l'autora per executar el present projecte.

### 1.3 Requisits previs

El projecte requereix coneixements previs de programació en *C++*[3] en crear un node de ROS, dels paquets *Behavior tree* [9], *YOLO* [30] i, del paquet *openpose* [10].

## Capítol 2

# INTRODUCCIÓ

La percepció és el procés d'extracció activa d'informació i d'elaboració de representacions. Els éssers humans la tenen fortament desenvolupada, però, és possible dotar a un robot d'aquesta percepció en un procés quotidià com en la realització d'un cafè? La resposta es troba en el projecte actual.

### 2.1 OBJECTIUS DEL PROJECTE

L'objectiu general del projecte és la implementació d'un mòdul de seguiment de les accions d'un humà i dels seus gestos en la preparació d'un cafè, per tal d'adaptar el comportament del robot. Per mitjà d'un mòdul cognitiu, el robot entendrà si pot continuar amb la seva tasca, quines tasques pot continuar fent en paral·lel o si es requereix la seva col·laboració per tal de dur a terme una tasca col·laborativa. Per complir l'objectiu general s'ha d'estudiar la viabilitat dels següents objectius específics:

- Estudi previ del paquet *Behavior Tree*, creació de l'arbre de comportament de la interacció humà-robot i la respectiva integració amb ROS.
- Estudi previ del paquet *YOLO*, definició dels objectes a detectar, preparació de la informació necessària, entrenament de la mateixa xarxa neuronal i la respectiva integració amb ROS.
- Estudi previ del paquet *OpenPose*, validació del mateix i la respectiva integració amb ROS.
- Desenvolupar l'aplicació per tal d'executar en temps real d'un mòdul cognitiu, utilitzant la detecció d'objectes, persones i, l'arbre de comportament.
- Provar que l'aplicació funcioni segons diferents casuístiques i documentar-ho.

### 2.2 ABAST DEL PROJECTE

L'abast del present projecte inclou el disseny d'una aplicació de percepció per visió *RGB* a excepció d'alguns inputs que seran expressats per senyals de la cafetera. Per tal de mantenir la versatilitat

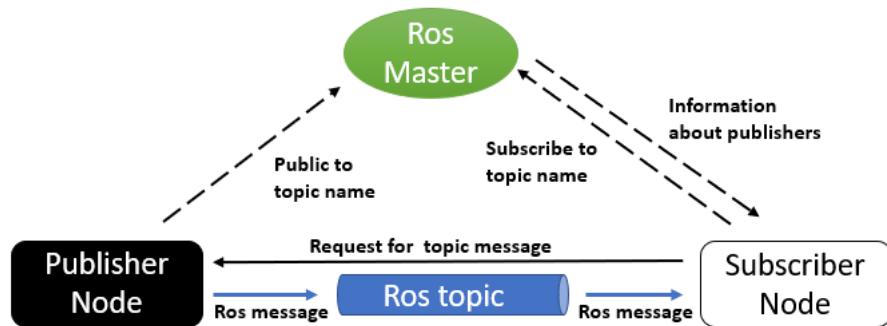
de l'aplicació s'exclou del present projecte tant la selecció del robot com el desenvolupament del seu comportament.

## Capítol 3

# CONEIXEMENTS PREVIS

### 3.1 ROS

El sistema operatiu de robòtica modular; Robotics Operating System [35] és una infraestructura de treball que permet fer un desenvolupament de forma estructurada, útil i efectiva, facilitant la programació i interconnexió entre les parts que l'aplicació ha de contenir. Es tracta d'un sistema què es divideix en conjunt de mòduls (nodes) que poden intercanviar informació a través de missatges. També, conté eines i llibreries que ajuden a obtenir, construir, escriure i executar codi entre diferents màquines. L'estruccura bàsica de *ROS* es representa a la figura 3.1.



Imatge 3.1: Exemple bàsic de ROS

A continuació s'explicaran les diferents estructures a utilitzar:

#### 3.1.1 Nodes

Els *nodes* de *ROS* [40] són bàsicament executables què utilitzen eines de *ROS* per a comunicar-se amb altres nodes mitjançant els tòpics, serveis i accions. Tal com és mostra en la figura 3.1, aquests poden ser publicadors (*Publisher Node*), o bé subscriptors (*Subscriber Node*). Els avantatges d'utilitzar nodes són els següents:

- Permet aïllar funcionalitats que estan dins d'un mateix context.

- La complexitat dels codis es redueix, ja que cada node tindrà el seu propi codi.
- Permet tenir diferents algoritmes i executar-los en cas que siguin necessaris de forma senzilla, al mateix *workspace*.

### 3.1.2 Packages (Paquets)

Els nodes es creen dins de paquets (*Packages*)<sup>[38]</sup>, que contenen els fitxers necessaris per definir el node. Aquests, requereixen els següents fitxers:

- **CMakeLists.txt**: és el fitxer d'entrada del sistema de compilació CMake bàsic el qual descriu com crear el codi i on instal·lar-lo.
- **package.xml**: conté la versió del paquet, el nom, la persona que duu a terme el manteniment i quines són les dependències que té amb altres paquets.

### 3.1.3 Tòpics

Els *tòpics de ROS* <sup>[41]</sup> són canals de comunicació unidireccional que permeten enviar informació d'un node (publicador) a un o més nodes (subscriptors) a una freqüència donada. No té cap mena de verificació sobre si la informació s'ha rebut correctament o no. La informació bàsica que s'envia és un missatge (msg). A la figura 3.1 s'observa com el *Publisher Node* reb un tòpic del *Subscriber Node* (línia negra) i, aquest, tanmateix, publica (o envia) un tòpic al *Subscriber Node* (línia blava). En els nodes, existeixen funcions anomenades *callbacks*, les quals reben la informació del tòpic on aquest està subscript.

### 3.1.4 Accions

Les Accions <sup>[36]</sup> apareixen per poder fer accions sota demanda, però l'execució d'aquestes accions es pot allargar en el temps. Com que no és viable mantenir el node sol·licitant bloquejat fins que l'acció acabi, aquest tipus de comunicació retorna immediatament quan s'ha enviat la sol·licitud (*goal*) i permet saber la seva evolució a través de missatges de Feedback i d'estat (*result*).

### 3.1.5 ROS Master

El *ROS Master* <sup>[39]</sup> és el cervell, què controla totes les diferents interconnexions (tòpics, serveis) entre els diferents nodes. En el cas de la figura 3.1 és l'encarregat de controlar la interconnexió entre els diferents tòpics.

### 3.1.6 Namespaces

Els Noms tenen un paper molt important en ROS: els nodes, tòpics i paràmetres tenen un nom. Cada llibreria de ROS suporta un *remapping*, què vol dir que un programa compilat es torna a

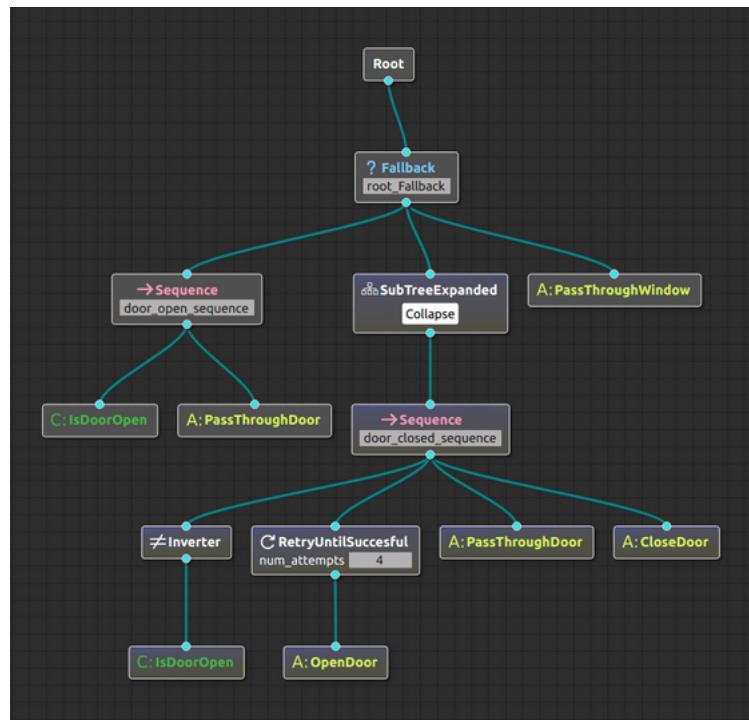
configurar durant l'execució per a ajustar-se a la topologia i assignació de noms dels nodes de cada sistema ROS.

## 3.2 Behavior Tree (Arbre de Comportament)

### 3.2.1 Introducció

Un arbre de comportament *Behavior Tree* [9] és una forma fàcil d'estructurar diferents tasques pròpies d'un procés així com els diferents canvis que inclouen. Un exemple d'arbre de comportament aplicat en obrir una porta és el que es mostra en la imatge 3.2.

La forma de lectura d'aquest arbre és de dalt a baix i d'esquerra a dreta. Bàsicament l'arbre primer és pregunta si la porta està oberta (*IsDoorOpen*), si és així, el robot passarà per la porta (*PassThroughDoor*). Si no és així, continuarà amb el subprocés *SubTreeExpanded*. Aquest subprocés primer mirarà si la porta està tancada (*inverter IsDoorOpen*), si és així provarà durant 4 cops d'obrir-la amb l'acció (*OpenDoor*). Una vegada la porta estigui oberta, el robot passarà per la porta (*PassThroughDoor*) i després tancarà la porta (*CloseDoor*). Si aquest subprocés falla, el robot passarà per la finestra *PassThroughWindow*.



Imatge 3.2: Exemple bàsic d'un arbre de comportament

Els principals avantatges són:

- Permeten compondre comportaments complexos que inclouen arbres sencers com a subprocessos d'un arbre més gran.

- Permet la seva representació gràfica, el qual facilita la seva lectura.
- Permet una personalització en la nomenclatura.
- Permet la reutilització d'alguns dels seus components.

### 3.2.2 Accions i Condicions

Les fulles de l'arbre de comportament estan formades per accions i condicions:

- **Accions:** és allò que s'ha de realitzar per complir un propòsit. Si s'observa la imatge 3.2, exemples serien les accions *PassThroughDoor*, *OpenDoor*, *PassThroughWindow* i *CloseDoor*.
- **Condicions:** és una clàusula lògica condicional que pot ser veritable o falsa. Si s'observa la imatge 3.2, un exemple seria la condició *IsDoorOpen*.

### 3.2.3 Tick

L'estat de l'arbre de comportament es pot conèixer fent-ne *tick*. A cada *tick* l'arbre s'actualitzarà i retornarà en quin estat es troba l'acció o condició en curs.

Quan en un node de l'arbre se li fa *tick*, aquest torna un *NodeStatus* (estat del node). Aquests poden ser:

- **Success:** informa que l'operació ha acabat i ha tingut èxit.
- **Failure:** informa que l'operació ha acabat i no ha tingut èxit.
- **Running:** ho retornen nodes asíncrons, quan la seva execució no ha estat acabada i, per tant, necessiten més temps per retornar un resultat vàlid.

### 3.2.4 Nodes

Els nodes són cadascuna de les tasques que controlen el flux de decisió i execució. Existeixen dos tipus de nodes:

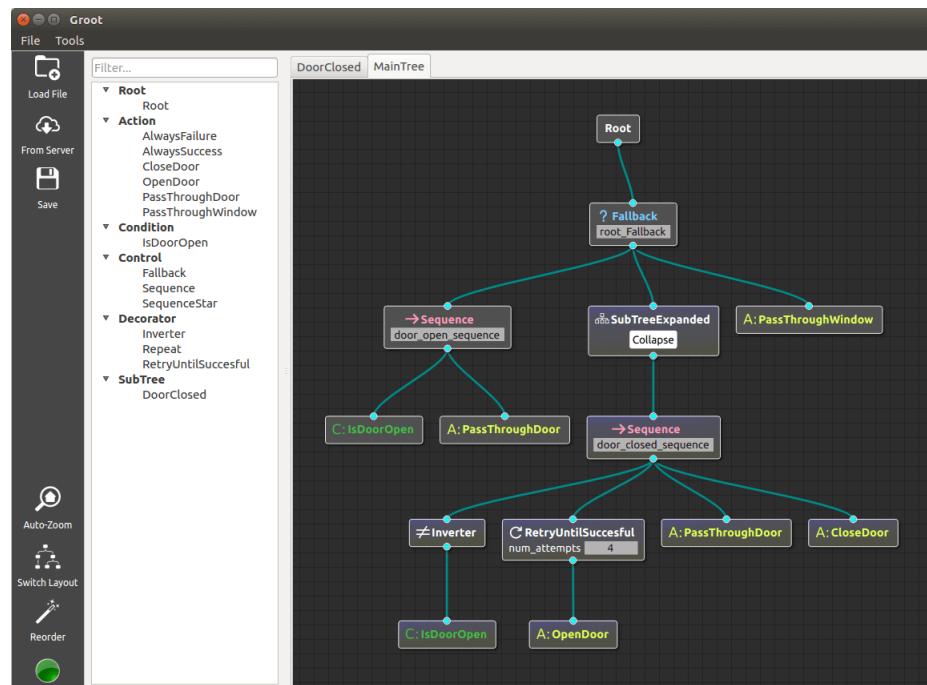
- **Control Nodes** (Nodes de Control): Aquests nodes poden contenir diverses accions o condicions ordenades per prioritat d'execució. Els nodes de control més utilitzats són els següents:
  - **Fallback** (Acció Alternativa): aquest node anirà executant seqüencialment les seves accions/condicions filles fins que una tingui èxit. Un exemple es troba en la imatge 3.2, si la condició *IsDoorOpen* falla, procedirà a l'execució del subprocés *SubTreeExpanded*.
  - **Sequence** (Seqüència): aquest node anirà executant seqüencialment les seves accions/-condicions filles mentre aquestes vagin tenint èxit. Per exemple, en la imatge 3.2 si la condició *IsDoorOpen* ha estat satisfactòria, procedirà amb l'acció *PassThroughDoor*.

- **Decorator Nodes** (Nodes decoradors): Aquests nodes només poden contenir una acció o condició. Els nodes decoradors més utilitzats són els següents:
  - **Timeout** (Temps d'espera): Aquest node esperarà un temps a què l'acció o condició s'efectuï. Si no és així, al següent *tick* retornarà un fracàs.
  - **Retry Until Successful** (Tornar-ho a provar fins que tingui èxit): Aquest node provarà un nombre d'iteracions a què l'acció o condició s'efectuï. Si no és així i, si al cap del nombre d'iteracions preestablertes l'acció o condició no ha tingut èxit, retornarà un fracàs. Un exemple es troba en la imatge 3.2, en la seqüència de *door\_closed\_sequence*, on s'aplica 4 intents a l'acció *OpenDoor*.
  - **Inverter** (Inversor): Al següent *tick* d'aquest node retornarà el resultat de l'acció o condició invertit. Un exemple es troba en la imatge 3.2, en la seqüència de *door\_closed\_sequence*, on s'inverteix la condició de *IsDoorOpen*.

### 3.2.5 Groot

Groot [7] és un editor gràfic, desenvolupat en C++[3] i Qt[6] i és compatible amb el *package BehaviorTree* [9].

L'aplicació permet crear i obrir arbres en format *XML*[8], canviar tot el *layout* de format horitzontal a vertical, reordenar de forma automàtica l'estructura i, veure si l'arbre està constituït de forma correcta o incorrecta (cercle verd/vermell inferior esquerre).



Imatge 3.3: Example bàsic d'aplicació del Groot

La imatge 3.3 mostra com l'aplicació permet crear accions, condicions i subprocessos de forma personalitzada així com l'existència dels nodes controladors i decoradors.

Finalment, aquesta interfície d'usuari permet arrosseggar accions, condicions, nodes i subprocessos a la zona de desenvolupament (Zona Negra), reutilitzant-les sempre i quant siguin necessàries.

Cada subprocés es desenvoluparà en una zona de treball específica. Tal com es mostra en la mateixa imatge hi ha una finestra *MainTree* la qual permet desenvolupar l'arbre general i, una altra *DoorClosed*, la qual permet desenvolupar el subprocés específic.

### 3.3 Càmera USB

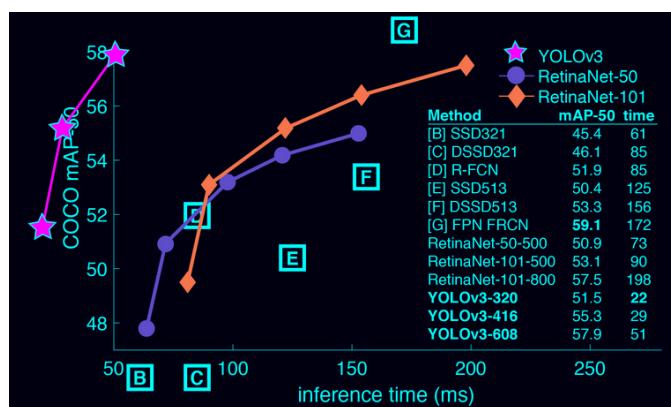
Per garantir una bona detecció de l'entorn s'utilitzarà la càmera USB *Astra Pro* [28]. La resolució d'aquesta càmera és de 1280x720, suficient per detectar la zona de treball. Tanmateix, conté ja drivers de ROS desenvolupats el qual facilita la posada en marxa i la integració amb ROS [37]. El procediment de calibració d'aquesta es pot observar en l'annex B.

### 3.4 YOLO - You Only Look Once

#### 3.4.1 Descripció

El *Yolo* (*You Only Look Once*) - Només mires una vegada [30] és un sistema de codi obert que permet la detecció d'objectes en temps real, el qual fa ús d'una única xarxa neuronal convolucional per a detectar objectes en imatges.

Si s'observa la imatge 3.4 és el detector que amb menys temps d'inferència, aconsegueix un *mAP* - *Mean Average Precision* (mitjana de precisió) millor.



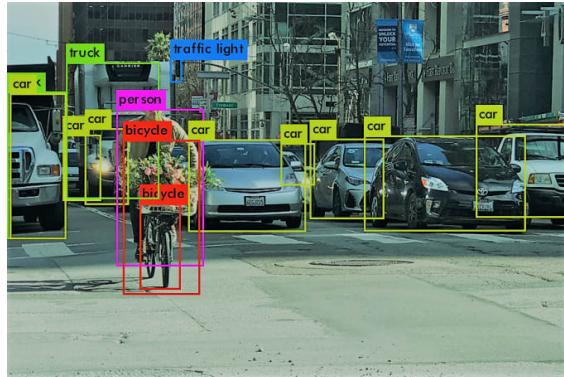
Imatge 3.4: Comparació amb altres detectors

Aquest detector aplica una única xarxa neuronal a la imatge completa. Aquesta xarxa divideix la imatge en regions, aplica el model en diverses ubicacions i escales prenent quadres de delimitació i probabilitats per a cada regió. Els sistemes de detecció prèvia reutilitzen classificadors o

localitzadors per realitzar la detecció.

Aquests quadres de delimitació es ponderen per les probabilitats previstes. Les regions amb puntuació alta de la imatge es consideren deteccions (vegeu imatge 3.5). L'algoritme aprèn representacions generalitzades dels objectes, permetent un baix error de detecció per a entrades noves, diferents del conjunt de dades d'entrenament.

Aquesta xarxa neuronal ve pre-entrenada per detectar un cert grup d'objectes, però, és fàcilment entrenable per detectar altres objectes segons l'aplicació desitjada.



Imatge 3.5: Resultat d'una imatge classificada amb Yolo

### 3.4.2 Darknet

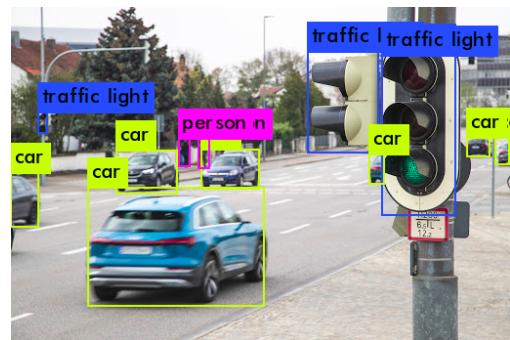
*Darknet* [31] és una xarxa neuronal de codi obert, escrita amb C [46] i CUDA [47]. Ràpida, fàcil d'instal·lar i suporta tant el càlcul amb una CPU com amb una GPU [15].

La mateixa xarxa permet descarregar un fitxer *.weights*[33], què conté pesos d'una xarxa pre-entrenada [32]. Una vegada descarregat el fitxer, permet detectar en qualsevol imatge un objecte que pertanyi a una de les classes amb què s'ha dut a terme el preentrenament.

En la imatge 3.6 és mostra el resultat de la detecció d'una xarxa preentrenada sobre una imatge qualsevol.



(a) Imatge sense tractar



(b) Imatge entrenada amb els pesos [33]

Imatge 3.6: Resultat de la detecció d'una xarxa preentrenada

### 3.4.3 Google Colab

Per entrenar una xarxa neuronal com la que s'explica en l'apartat anterior 3.4.2, requereix un sistema de computació complex que contingui com a mínim una *GPU* (*Graphics Processing Unit*) [15] sempre que es vulgui obtenir càlculs ràpids i efectius. L'accés a aquest tipus de *hardware* és molt costós econòmicament, fet que pocs usuaris tenen el poder d'adquirir-ho.

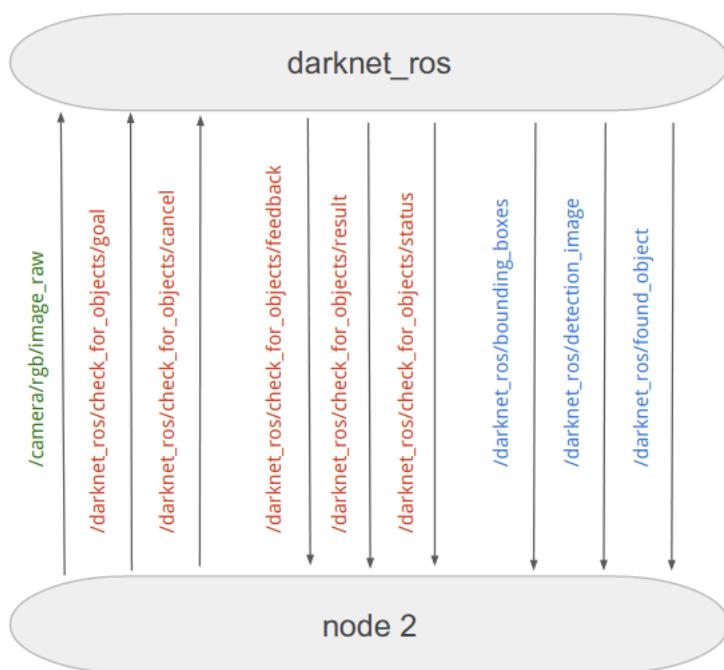
És per això que *google colab*[5] té un servei per subscripció que permet l'accés a una GPU. Aquest servei és altament accessible, fet que facilita l'entrenament de xarxes neuronals.

Aquest servei permet connectar el bloc de notes de *Colab* al *Google Drive* [11] de l'usuari, descarregar el paquet [31] i, preparar tota l'estructura necessària per entrenar aquesta xarxa neuronal.

### 3.4.4 Darknet ROS

Per tal que la integració amb ROS [35] sigui efectiva, és necessari tenir un paquet que tradueixi l'estructura de *YOLO* [30] a l'estructura amb què treballa ROS.

El paquet *darknet\_ros* [4] és un paquet de ROS desenvolupat per la detecció d'objectes en imatges de càmera. Aquest permet tant afegir un model preentrenat com afegir un nou model amb els mateixos objectes de detecció.



Imatge 3.7: Arquitectura del node *darknet\_ros*

Conté un node de ROS nomenat *darknet\_ros* i diversos tòpics i accions, tal com es mostra en la imatge 3.7. Aquests, es poden definir com:

- Tòpic */camera/rgb/image\_raw*: És la imatge d'entrada que utilitzarà l'algoritme de Yolo

per detectar objectes prèviament entrenats.

- **Acció /check\_for\_objects:** Aquesta acció [3.1.4] és l'encarregada de buscar objectes específics en la detecció de l'algoritme *Yolo* [3.4]. Aquesta està composta pels següents tòpics:
  - *goal*: està compost pel missatge *darknet\_ros\_msgs/CheckForObjectsActionGoal* (vegeu A.1) el qual permet enviar a l'algoritme una imatge per tal de realitzar la detecció.
  - *cancel*: està compost pel missatge *actionlib\_msgs/GoalID* (vegeu A.1) el qual permet cancel·lar el goal prèviament enviat.
  - *feedback*: està compost pel missatge *darknet\_ros\_msgs/CheckForObjectsActionFeedback* (vegeu A.1) el qual permet obtenir resposta en tot moment de com s'està processant el *goal* prèviament enviat.
  - *result*: està compost pel missatge *darknet\_ros\_msgs/CheckForObjectsActionResult* (vegeu A.1) el qual conté el resultat del *goal* prèviament enviat.
  - *status*: està compost pel missatge *actionlib\_msgs/GoalStatusArray* (vegeu A.1) el qual permet saber quin és l'estat del *goal* prèviament enviat.
- **Tòpic /bounding\_boxes:** Aquest tòpic dóna a conèixer els *Bounding boxes* o, les caixes que envolten els objectes detectats. Aquestes caixes es mostren les coordenades [ $x_{min}, y_{min}, x_{max}, y_{max}$ ] de l'objecte de detecció en la imatge de detecció. Tanmateix, proveeixen de la probabilitat de la detecció. Està format pel missatge *darknet\_ros\_msgs/BoundingBoxes* (vegeu annex A.1).
- **Tòpic /detection\_image:** És la imatge de sortida que conté els objectes detectats per la xarxa neuronal proveïda.
- **Tòpic /found\_object:** Aquest tòpic mostra un nombre que indica quants objectes s'han detectat a la imatge.

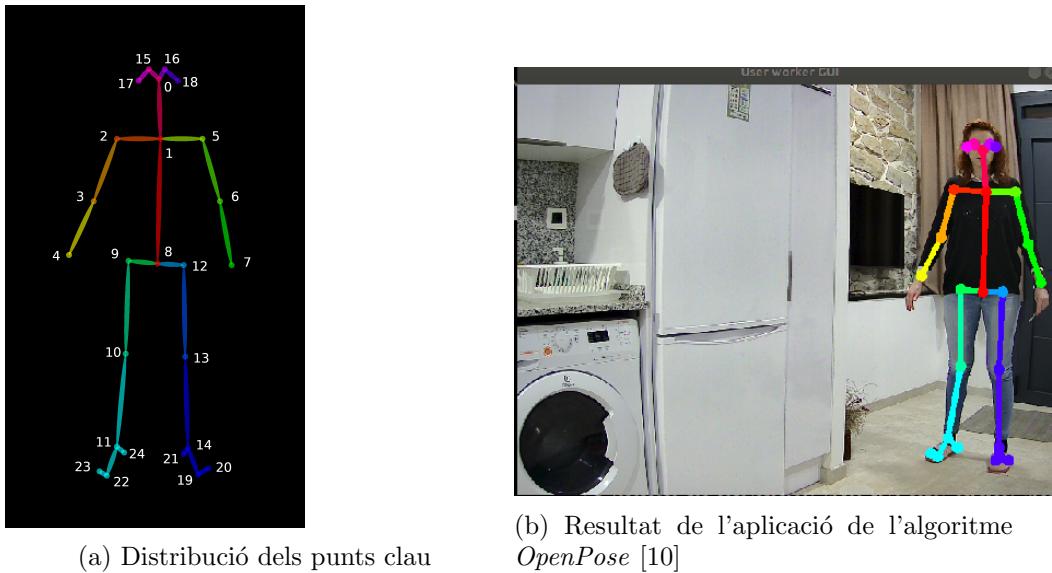
## 3.5 OpenPose

### 3.5.1 Descripció

L'*OpenPose* [10] és el primer sistema de detecció de diverses persones en temps real que detecta conjuntament els punts clau del cos humà, les mans, la cara i els peus. Permet detectar en total 135 punts clau en imatges individuals.

#### Punts Clau Cos

El paquet permet fins a la detecció de 25 punts claus del cos. Distribuïts i numerats tal com es mostra en la imatge 3.8a. Aquests formen part dels ulls, el nas, les articulacions del coll, dels braços, maluc, cames i peus. Si s'aplica l'algoritme a una imatge qualsevol, el resultat es mostra en la imatge 3.8b.



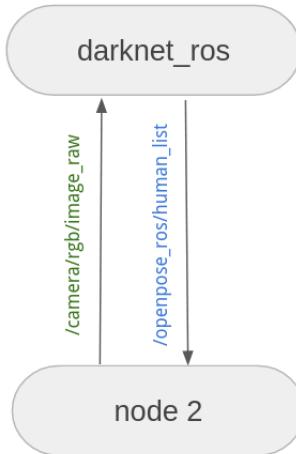
Imatge 3.8: Resultat de la detecció de l'algoritme openpose

### 3.5.2 OpenPose ROS

Per tal que la integració amb ROS [35] sigui efectiva, és necessari tenir un paquet que tradueixi l'estructura d'*OpenPose* [10] a l'estructura amb què treballa ROS.

El paquet *openpose\_ros* [48] és un paquet de ROS desenvolupat que utilitza la llibreria *OpenPose* (vegeu 3.5.1) en la detecció de persones en imatges d'una càmera.

Conté un node de ROS nomenat *openpose\_ros*. La distribució dels tòpics és mostra en la imatge 3.9.



Imatge 3.9: Arquitectura del node openpose\_ros

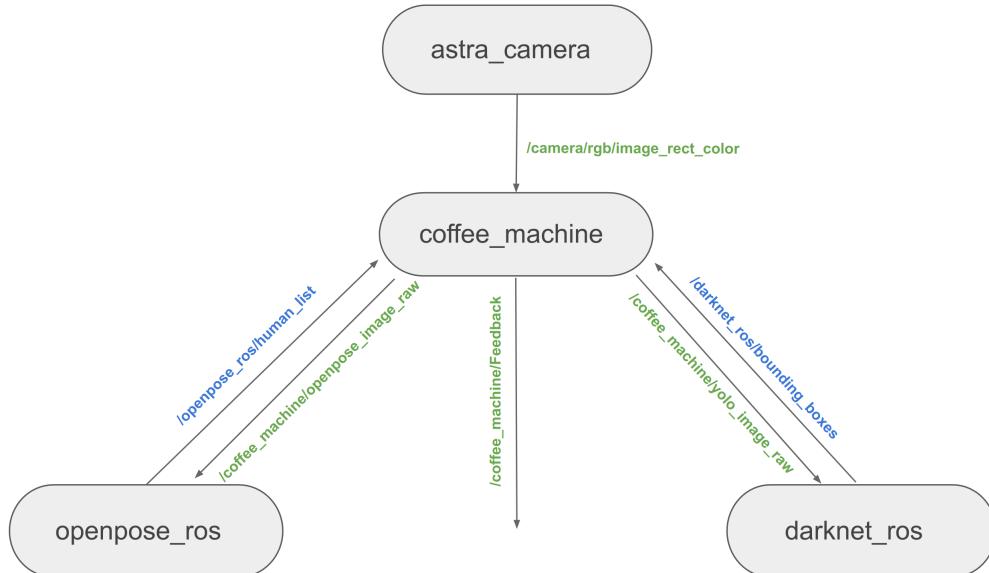
- **Tòpic /camera/rgb/image\_raw:** És la imatge d'entrada que utilitzarà l'algoritme *OpenPose* per detectar els punts claus de persones, cares i mans.

- **Tòpic openpose\_ros/human\_list.** Aquest tòpic el conforma el missatge *openpose\_ros\_msgs/OpenPoseHumanList* (Vegeu annex A.2). Aquest tòpic permet, per cada persona detectada:
  - Conèixer les *Bounding Boxes* o, caixes que envolten el seu cos.
  - Conèixer les *Bounding Boxes* o, caixes que envolten la cara.
  - Conèixer els punts clau del cos, obtenint la respectiva posició amb píxels i la probabilitat en la detecció.
  - Conèixer els punts clau de la cara, obtenint la respectiva posició amb píxels i la probabilitat en la detecció.
  - Conèixer els punts clau de la mà dreta, obtenint la respectiva posició amb píxels i la probabilitat en la detecció.
  - Conèixer els punts clau de la mà esquerra, obtenint la respectiva posició amb píxels i la probabilitat en la detecció.

### 3.6 Arquitectura de ROS

L’arquitectura de ROS amb què es treballa és mostra en la imatge 3.10. S’utilitzen diversos tòpics del node *openpose\_ros* (vegeu apartat 3.5.2), del node *darknet\_ros* (vegeu apartat 3.4.4) i, del node de la càmera *astra* (vegeu apartat 3.3).

El node *coffee\_machine* [24] és el node d’interconnexió entre els diferents paquets. Aquest paquet conté l’arbre de comportament de l’aplicació a desenvolupar. Bàsicament, és qui fa possible la seva execució i la detecció de les diferents accions en la realització d’un cafè.



Imatge 3.10: Arquitectura del ROS a desenvolupar

## Part II

# PART II

## Capítol 4

# IDENTIFICACIÓ I CREACIÓ DE L'ENTORN DE TREBALL:

### 4.1 Aplicació a desenvolupar:

#### 4.1.1 Introducció:

L'aplicació es basa en la interacció humà-robot en el procés de crear un cafè mitjançant una màquina de cafè en gra. Bàsicament, en la implementació d'un mòdul de seguiment de les accions d'un humà i reconeixement dels seus gestos, per tal d'adaptar el comportament del robot.



imatge 4.1: Escenari de treball

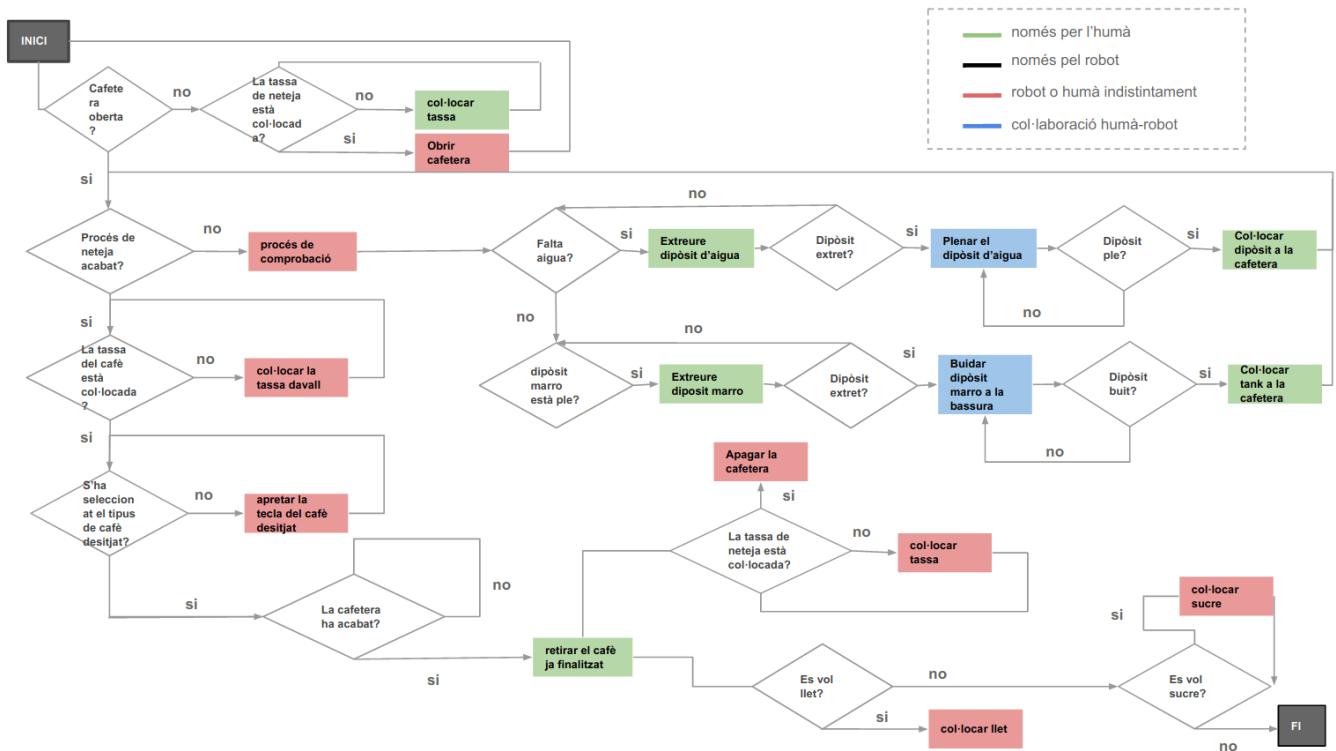
L'escenari de treball és el que és mostra en la imatge 4.1. Aquest està conformat pels següents objectes o *classes*:

- Dipòsit d'aigua

- Dipòsit contenidor de marro
- Tassa
- Cafetera [2]. Conté senyals lluminosos i una interície web què permeten mostrar estats i errors.
- Llet
- Sucre

#### 4.1.2 Arquitectura:

El diagrama de flux que regeix l'aplicació a desenvolupar pot observar-se en la imatge 4.2 i està compost dels següents subprocessos:



Imatge 4.2: Diagrama de flux

- Primerament, es comprovarà si la cafetera està encesa mitjançant el LED corresponent a la cafetera o bé el robot comunicant-se amb la cafetera intel·ligent. Si la cafetera no està oberta, es comprovarà si la tassa de neteja està col·locada; Si és així, l'humà o el robot procediran a obrir la cafetera. Si no és així, l'humà procedirà amb la col·locació de la tassa de neteja.
- Una vegada la cafetera estigui oberta, procedirà a executar un procés d'autoneteja. Si la cafetera no l'ha pogut complir, el robot o l'humà començaran un procés de comprovació.

Aquest procés consta d'una comprovació si falta aigua mirant el LED corresponent a la cafetera o bé mitjançant la comunicació del robot amb la cafetera intel·ligent. Si és així, l'humà extraurà el dipòsit d'aigua. Una vegada el dipòsit estigui extret, hi haurà una col·laboració humà-robot per tal d'omplir el dipòsit d'aigua i, finalment, una vegada el dipòsit estigui ple, l'humà col·locarà el dipòsit a la cafetera. Seguidament, i, una vegada acabat el procés de comprovació d'aigua mitjançant el LED indicador de la cafetera o bé el robot comunicant-se amb la cafetera intel·ligent, l'humà o el robot comprovaran si el dipòsit de marro està ple; Si és així, l'humà extraurà el dipòsit de marro. Una vegada el dipòsit estigui extret, es requerirà una col·laboració humà-robot per tal de buidar el dipòsit de marro a la brossa i, finalment, una vegada el dipòsit estigui buit, l'humà procedirà a col·locar-lo a la cafetera.

- Una vegada el procés de neteja hagi acabat, l'humà o el robot reemplaçaran la tassa de neteja per la tassa del cafè.
- Finalment, i, sempre que no s'hagi seleccionat el tipus de cafè desitjat l'humà o el robot procediran a pressionar la tecla del cafè requerit o enviar l'ordre corresponent a través de la interície web. Quan la cafetera hagi acabat, l'humà retirarà el cafè ja finalitzat. Aquí s'iniciarà un paral·lelisme; Per una banda, si la tassa de neteja no està col·locada, l'humà o el robot l'hauran de col·locar i, una vegada hi estigui, o un o l'altre haurà de procedir a apagar la cafetera. Per l'altra banda, si l'usuari final vol llet i/o sucre, l'humà o el robot hi hauran de col·locar llet i/o sucre. En aquest projecte s'ha decidit dotar d'un comportament elegant al robot i aquest esperarà que l'humà prengui la iniciativa escollint una de les tasques paral·leles, executant el robot la tasca que hagi deixat per fer l'humà. No obstant això, si l'humà no es decideix, passat prou temps el robot escollirà una de les tasques pendent a fer.

## 4.2 Clustering i detecció:

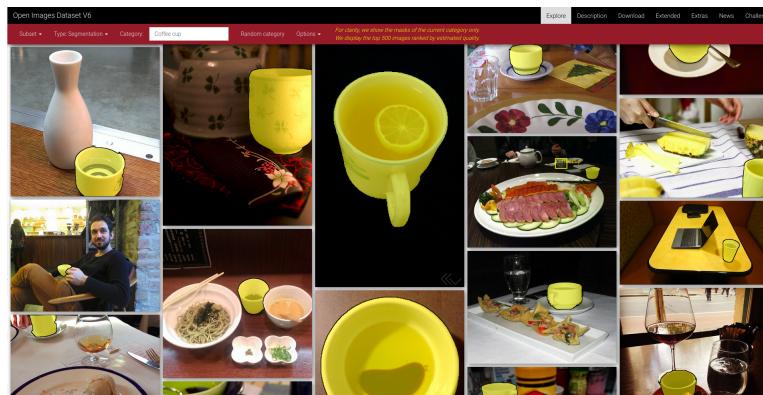
Per tal d'identificar l'entorn de treball és necessari entrenar la xarxa neuronal *darknet* [3.4.2] perquè sigui capaç de detectar els objectes (classes) esmentats en l'apartat 4.1.1.

Si es vol que l'entrenament s'efectuï amb poc temps i de forma eficaç, és necessari accedir a una GPU. És per això que es procedeix a entrenar aquesta xarxa neuronal amb l'eina *colab* 3.4.3.

### 4.2.1 Entrenament amb google colab

#### PREPARACIÓ DATASET:

Per tenir èxit en l'entrenament d'aquesta xarxa neuronal cal preparar un conjunt de dades (*dataset*) per a cada classe. Aquest *dataset* estarà format per **Imatges** i, cada imatge tindrà associat un document etiqueta **\*.txt**.



Imatge 4.3: *Open Images Dataset d'una tassa de cafè*

Classe	Nom OID	N. Im. Entrenament	N. Im. Prova
Llet	<i>Milk</i>	156	40
Cafè	<i>Coffee</i>	200	100
Màquina de Cafè	<i>CoffeeMaker</i>	200	55
Tassa	<i>Cup of Coffee</i>	200	100

Taula 4.1: *Relació classe amb el nom a OID*

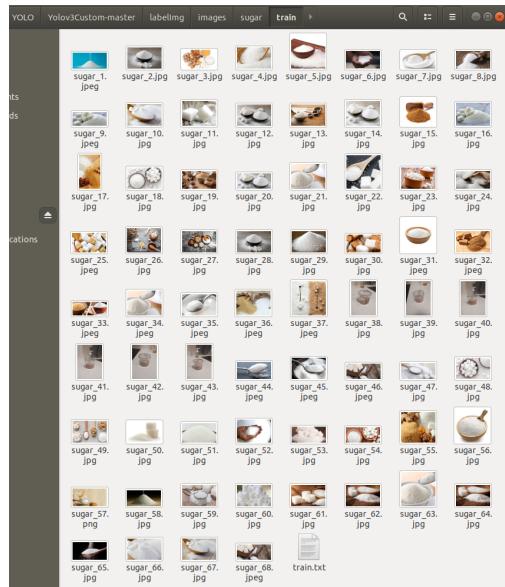
**Imatges:** Per cada classe, cal tenir un bon nombre d'imatges d'entrenament. *Google Open Source* [12] conté una base de dades anomenada *Open Images Dataset* [13]. Aquesta, permet obtenir *datasets* de classes, tal com es pot observar en la imatge 4.3

Fent referència a les classes que formen l'escenari de treball (vegeu 4.1.1), en la taula 4.1 es mostren aquelles que s'han descarregat de la base de dades *Open Images Dataset* [13]. A la mateixa taula i, per cada classe, s'observa el nom clau de cerca (*Nom OID*), el nombre d'imatges descarregades per dur a terme l'entrenament (*N. Im. Entrenament*) i, el nombre d'imatges descarregades per dur a terme proves internes. *N. Im. Prova*. Aquestes imatges de prova no poden formar part de les imatges d'entrenament.

Les classes no incloses dins del dataset (OID)[13], han estat creades pròpiament, fent una cerca exhaustiva d'imatges, tal com es mostra en la imatge 4.4.

**Etiquetes (Labels) .txt:** En cada imatge poden haver-hi tant diferents classes com diferents objectes de la mateixa classe. És per això que és necessària la creació d'un document etiqueta que identifiqui el següent:

- Quins objectes existeixen.
- De quina classe formen part.
- On estan ubicats dins de la mateixa imatge. Això es defineix a quina *Bounding box* o, caixa estan continguts dins la imatge.



(a) Dataset d'entrenament de la classe sucre



(b) Dataset de prova de la classe sucre

Imatge 4.4: Creació del dataset per a la classe sucre

En la imatge 4.5 s'observa el contingut del document. Cada línia consta d'un objecte: identificador, valor x i y del centre de la *Bounding Box* i, finalment, el valor de l'amplada i de l'altura de la *Bounding Box*. Tots els valors han d'estar normalitzats, si no és així, l'entrenament de l'algoritme no pot efectuar-se.

El propi dataset (OID)[13] permet descarregar aquest document per cada imatge, però, aquests valors no estan normalitzats. Hi ha dues formes de normalitzar aquests documents:

- (Opció A): creació d'un propi algoritme que apliqui la normalització a cadascun dels objectes detectats.
- (Opció B): creació d'un nou document *.txt* normalitzat a partir d'una interfície d'usuari dedicada.

real95.txt				
	CLASSE_ID	Y_CENTRE_NORM	AMPLADA_NORM	ALTURA_NORM
4	0.406629	0.627852	0.060584	0.090304
3	0.558446	0.650190	0.091946	0.085551
1	0.564861	0.629753	0.057733	0.035171
2	0.560941	0.535646	0.189594	0.403992
0	0.724163	0.623574	0.066999	0.108365

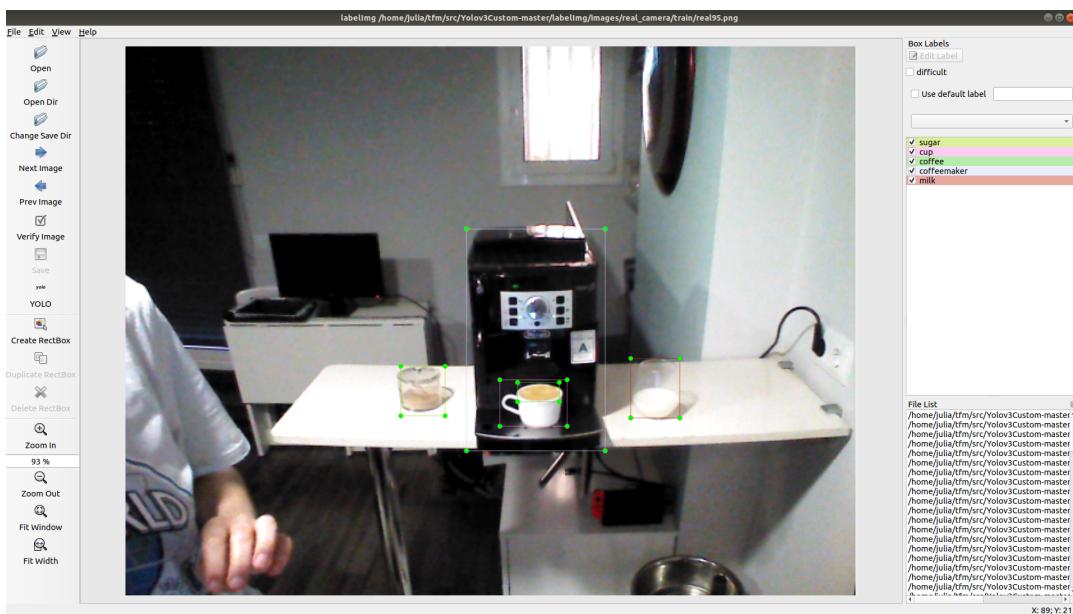
Imatge 4.5: Document etiqueta de la imatge 4.6

Classe	ID
Llet	0
Cafè	1
Màquina de cafè	2
Tassa	3
Sucre	4
Dipòsit Marro	5
Dipòsit Aigua	6

Taula 4.2: Relació classe amb l'ID assignada

**Creació dels .txt etiqueta:** La forma de normalitzar escollida és la B, creació d'un nou document *.txt* normalitzat a partir d'una interfície d'usuari dedicada. Per això, s'utilitza el paquet *labelImg* [42]. Aquest paquet proveeix d'una interfície d'usuari (vegeu imatge 4.6) la qual permet establir un directori d'entrada proveïdor d'imatges i un de sortida generador d'etiquetes *label*, permet crear rectangles *Bounding Boxes* i assignar-los a una classe específica.

Cal remarcar la importància que cada classe tingui assignat un ID estàtic. La taula 4.2 mostra el nom de la classe amb l'ID assignada. Aquest ID és el que s'assignarà sempre a la classe específica i és el que utilitzarà el mateix algoritme *Yolo 3.4* per referir-se a la classe.

Imatge 4.6: Interfície d'usuari *LabelImg*

## ENTRENAMENT AMB GOOGLE COLAB:

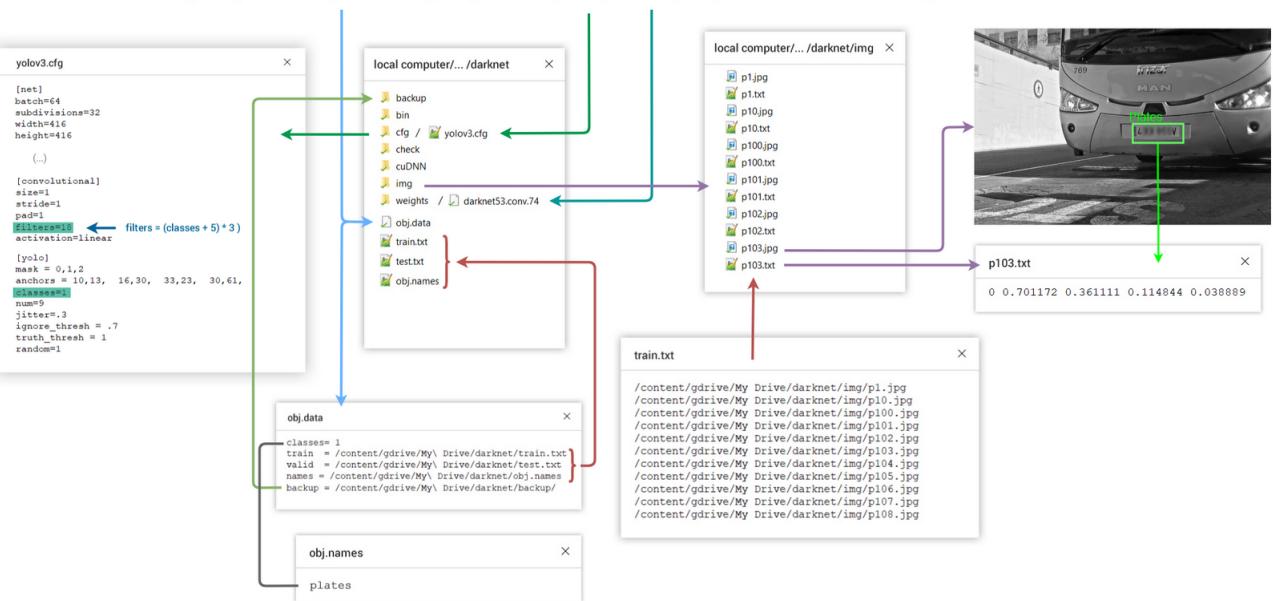
L'entrenament de *Google Colab* (vegeu 3.4.3) es realitza segons la pàgina [23]. Aquesta permet connectar un usuari *drive* amb els serveis de *Google Colab*. Tanmateix, permet descarregar el mateix paquet *darknet 3.4.2*, habilitar l'execució per la *GPU*, compilar-lo i executar l'entrenament.

Per això, és necessari crear una carpeta a *drive* pròpia de l'usuari [20] seguint la imatge 4.7 . Aquesta consta de les següents parts:

- **backup**: És el directori on s'aniran guardant els pesos en cada entrenament.
- **cfg**: És el directori que conté l'arxiu de configuració de la xarxa *.cfg* convolucionals adaptat pel nombre de classes a entrenar.
- **img**: És el directori que conté totes les imatges i els arxius etiqueta respectius.
- **weights**: Conté un arxiu amb un pes inicial per entrenar l'algoritme amb dades personalitzades.
- **train.txt**: És l'arxiu què conté la localització de totes les imatges a entrenar.
- **test.txt**: És l'arxiu què conté la localització de totes les imatges a provar.
- **obj.names**: És l'arxiu que enllaça el nom de la classe amb l'ID, cada línia és una classe, la primera serà la 0, *milk* i l'última la 6, *water\_tank*.
- **obj.data**: És l'arxiu què indica el nombre de classes a entrenar i, on es poden localitzar els arxius *train.txt*, *test.txt*, *obj.names* i, el directori *backup* dins de la carpeta en qüestió.

## Darknet. YOLOv3 Configuration files on colab notebook

```
!./darknet detector train "/content/gdrive/My Drive/darknet/obj.data" "/content/gdrive/My Drive/darknet/yolov3.cfg" "/content/gdrive/My Drive/darknet/darknet53.conv.74" -dont_show
```



Imatge 4.7: Estructura del directori *darknet* propi

#### 4.2.2 Integració amb ROS

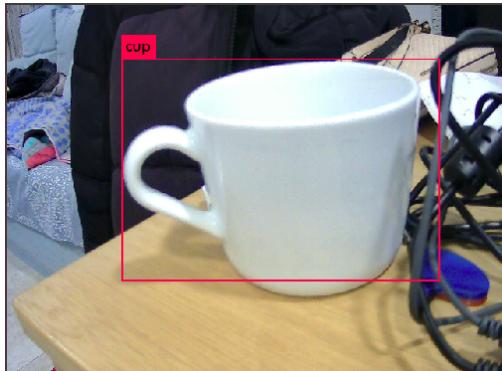
Una vegada acabat l'entrenament, cal fusionar el resultat de l'entrenament amb el paquet darknet\_ros (vegeu 3.4.4).

Per això cal seguir els següents passos dins del paquet [4]:

1. Afegir dins de la carpeta **cfg**, l'arxiu bàsic de configuració de yolo *yolov3.yaml* [16].
2. Afegir dins de la carpeta **yolo\_network\_config\cfg** l'arxiu *yolov3-coffee-machine\_7\_classes.cfg* [17].
3. Afegir dins de la carpeta **yolo\_network\_config \weights** l'arxiu *yolov3-coffee-machine\_7\_classes\_final.weights* [18].
4. Afegir dins la carpeta **launch** els arxius *darknet\_ros.launch* i, l'arxiu *yolo\_v3.launch* [19].
5. Finalment, si és necessari, executar en una terminal l'arxiu *yolo\_v3.launch*.

#### 4.2.3 Resultat

El resultat és la detecció de les classes prèviament entrenades. A la imatge 4.8a es pot observar el *Bounding Box* resultat de la detecció d'una tassa i, conseqüentment, a la imatge 4.8b es pot observar la resposta del tòpic *darknet\_ros/bounding\_boxes*.



(a) Resultat de la detecció de l'algoritme Yolo

```
header:
  seq: 23
  stamp:
    secs: 1606436760
    nsecs: 694015951
  frame_id: "detection"
image_header:
  seq: 65568
  stamp:
    secs: 1606436746
    nsecs: 327758296
  frame_id: "usb_cam"
bounding_boxes:
  -
    probability: 0.998476326466
    xmin: 147
    ymin: 68
    xmax: 557
    ymax: 354
    id: 3
    Class: "cup"
```

(b) Resultat de la detecció del tòpic *darknet\_ros/bounding\_boxes*

Imatge 4.8: Resultat de la detecció utilitzant el paquet *darknet\_ros*

## Capítol 5

# IDENTIFICACIÓ I CREACIÓ DEL COMPORTAMENT HUMÀ:

### 5.1 Accions i condicions a detectar:

Una vegada vist com identificar l'entorn de treball, cal ser capaç de distingir el comportament humà. En l'apartat 4.1.2 s'ha esmentat que hi haurà accions efectuades per l'humà, fet que el robot ha de ser capaç de detectar-les, identificar si l'acció s'ha realitzat de forma satisfactòria i, procedir amb la següent acció. El fet que el robot estigui dotat d'un comportament elegant, fa que aquest esperi que l'humà prengui la iniciativa escollint una de les tasques paral·leles, executant el robot la tasca que hagi deixat per fer l'humà. No obstant això, si l'humà no es decideix passat prou temps, el robot escollirà una de les tasques pendents a fer.

En la imatge 4.2 s'especifica quines són les accions i condicions condicionades únicament per l'humà, en color *verd* i, les que poden desenvolupar paral·lelament, en color *vermell*. Hi ha dues formes de detectar si aquestes accions o condicions s'han desenvolupat de forma satisfactòria: amb el desenvolupament d'un algoritme de percepció o, mitjançant la respectiva habilitació per un senyal extern. Aquest senyal extern serà definit per la detecció del LED corresponent a la cafetera o bé pel robot comunicant-se amb la cafetera intel·ligent.

En funció de l'acció o condició s'utilitzarà una forma específica, tal com es mostra en la taula 5.1.

Les següents parts del present capítol fan referència a la creació de l'algoritme de percepció que utilitzaran algunes de les accions i condicions.

### 5.2 Setup de l'algoritme OpenPose

Per tal de detectar la posició de l'humà en cadascuna d'aquestes accions o condicions és necessari l'ús del paquet *open\_pose* [3.5]. Mitjançant l'ús d'aquest paquet s'extrauran els 25 punts clau del

Nom	Acció o Condició	Forma de detecció
Està la cafetera oberta?	condició	habilitada per senyal exterior
Col·locar la tassa de neteja a la cafetera	acció	algoritme percepció
Encendre la cafetera	acció	habilitada per senyal exterior
El procés de neteja ha acabat?	condició	habilitada per senyal exterior
Hi ha prou aigua en la cafetera?	condició	habilitada per senyal exterior
El dipòsit d'aigua està ple?	condició	habilitada per senyal exterior
S'ha extret el dipòsit d'aigua de la cafetera?	condició	algoritme percepció
Omplir el dipòsit d'aigua de la cafetera	acció	habilitada per senyal exterior
S'ha col·locat el dipòsit d'aigua a la cafetera?	condició	algoritme percepció
El dipòsit de marro està buit?	condició	habilitada per senyal exterior
El dipòsit de marro està ple?	condició	habilitada per senyal exterior
S'ha extret el dipòsit de marro de la cafetera?	condició	algoritme percepció
Buidar el dipòsit de marro de la cafetera	acció	habilitada per senyal exterior
S'ha col·locat el dipòsit de marro en la cafetera?	condició	algoritme percepció
S'ha seleccionat el café desitjat?	condició	habilitada per senyal exterior
Prémer el botó del café desitjat	acció	habilitada per senyal exterior
S'ha retirat la tassa de cafè de la cafetera?	condició	algoritme percepció
S'ha col·locat la tassa de neteja a la cafetera?	condició	algoritme percepció
Col·locar tassa de café a la cafetera	acció	algoritme percepció
El café està llest?	condició	habilitada per senyal exterior
Apagar la cafetera	acció	habilitada per senyal exterior
Es vol llet?	condició	habilitada per senyal exterior
Es vol sucre?	condició	habilitada per senyal exterior
S'ha afegit llet?	condició	algoritme percepció
S'ha afegit sucre?	condició	algoritme percepció
Afegir llet	acció	habilitada per senyal exterior
Afegir sucre	acció	habilitada per senyal exterior

Taula 5.1: Relació d'accions i condicions amb la forma de detecció

cos humà (vegeu la imatge 3.8a), dels quals solament s'utilitzaran els següents:

- Punt clau **numero 4** - articulació canell dret.
- Punt clau **numero 7** - articulació canell esquerra.

## 5.3 Creació del paquet `coffe_machine`

El paquet *coffe\_machine* [24] és un node de ROS exclusivament creat per l'aplicació a desenvolupar. És el *cervell* que efectua la interconnexió de tots els nodes implicats, conté la lògica de l'aplicació i proveeix al sistema de les noves funcionalitats. Aquest paquet, així com l'arquitectura d'interconnexió amb els altres nodes de ROS es pot observar en l'apartat 3.6.

### 5.3.1 Integració d'OpenPose

Per tal de la correcta integració d'*OpenPose* amb ROS, s'utilitza el paquet *openpose\_ros* (vegeu l'apartat 3.5.2).

Amb la imatge obtinguda de la càmera USB (vegeu l'apartat 3.3), aquesta es publica al tòpic */coffe\_machine/openpose\_image\_raw*. Una vegada l'algoritme de detecció detecti els 25 punts claus, el tòpic *openpose\_ros/human\_list* retornarà la informació d'aquests punts claus (vegeu imatge 3.10).

**find\_joint** El node *coffe\_machine* conté dins la classe *CoffeMachineROSNode* una funció anomenada *find\_joint*. Donat un missatge del tòpic *openpose\_ros/human\_list* i el número de l'articulació (4 o 7 en aquest cas), aquest retorna un missatge tipus *openpose\_ros\_msgs/PointWithProb* (vegeu A.2), el qual conté la x i y de les coordenades de la imatge on es troba l'articulació i la probabilitat de detecció d'aquesta articulació.

### 5.3.2 Integració del YOLO

Una vegada obtingut el resultat del YOLO (vegeu apartat 4.2.3), cal veure si aquest resultat de detecció conté l'objecte que defineix l'acció o condició en qüestió, els quals tenen un objecte clau a detectar. La taula 5.2 mostra quin és l'objecte a detectar en funció de l'acció o condició a detectar.

Amb la imatge obtinguda de la càmera USB (vegeu l'apartat 3.3), aquesta es publica al tòpic */coffe\_machine/yolo\_image\_raw*. Una vegada l'algoritme de detecció detecti els objectes, el tòpic *darknet\_ros/bounding\_boxes* retornarà les *Bounding Boxes* explicades en l'apartat 4.2.3 amb la informació de la localització d'aquests objectes en la imatge (vegeu imatge 3.10).

**find\_class** El node *coffe\_machine* conté en la classe *CoffeMachineROSNode* una funció anomenada **find\_class**. Donat un missatge del tòpic *darknet\_ros/bounding\_boxes* i el nom d'una classe a detectar, aquest retorna un missatge tipus *darknet\_ros\_msgs/BoundingBox* (vegeu A.1) de la classe

Acció o condició	Objecte Clau a detectar
Col·locar la tassa de neteja a la cafetera	<i>cup,coffeemaker</i>
S'ha extret el dipòsit d'aigua de la cafetera?	<i>water_tank</i>
S'ha col·locat el dipòsit d'aigua a la cafetera?	<i>water_tank</i>
S'ha extret el dipòsit de marro de la cafetera?	<i>marro_tank</i>
S'ha col·locat el dipòsit de marro a la cafetera?	<i>marro_tank</i>
S'ha retirat la tassa de café de la cafetera?	<i>cup,coffee</i>
S'ha col·locat la tassa de neteja a la cafetera?	<i>cup</i>
Col·locar tassa de café	<i>cup,coffeemaker</i>
S'ha afegit llet?	<i>milk</i>
S'ha afegit sucre?	<i>sugar</i>

Taula 5.2: *Relació d'accions i condicions amb les classes a detectar*

requerida, sempre que aquesta existeixi dins de la imatge. Aquest missatge conté la  $x_{min}, x_{max}, y_{min}$ ,  $y_{max}$  del rectangle que conté la classe en píxels.

Hi ha una altra opció implementada per a obtenir *Bounding Boxes* en imatges, la qual es basa en publicar la imatge i un *id* d'objecte al tòpic */darknet\_ros/check\_for\_objects/goal*, aquest es basa en un missatge tipus *darknet\_ros\_msgs/CheckForObjectsActionGoal* (vegeu A.1). Mentre aquesta acció no s'hagi executat, permet saber en tot moment l'estat de l'*action goal* amb el tòpic */darknet\_ros/check\_for\_objects/Feedback* i, conèixer el resultat una vegada s'obtingui amb el tòpic */darknet\_ros/check\_for\_objects/result* (vegeu A.1).

### 5.3.3 Algoritme per la detecció de la subjecció d'objectes

Una vegada obtingut la localització del canell esquerre, del canell dret i la localització de l'objecte clau, cal comprovar si l'humà està agafant l'objecte clau en qüestió. Per això cal mesurar la distància entre les coordenades del canell i les del centre de la *Bounding Box* que conté l'objecte clau, de la següent forma:

$$\begin{aligned} \text{fabs}(x - 0.5 * (xmin+xmax)) &< (0.5 * (xmax-xmin) + dist) \text{ and} \\ \text{fabs}(y - 0.5 * (ymin+ymax)) &< (0.5 * (ymax-ymin) + dist) \end{aligned}$$

1. En primer lloc,  $x_{min}$  i  $x_{max}$  són les posicions mínima i màxima en x de les posicions del perímetre del rectangle que emmarca l'objecte. Es calcula el centre de la *Bounding Box* fent la mitja entre la respectiva  $x_{min}$  i la  $x_{max}$  ( $0.5 * (x_{min} + x_{max})$ ).
2. Seguidament, s'efectua valor absolut de la diferència de la posició de x del canell amb el càlcul anterior, obtenint quina distància té el canell respecte al centre de la *Bounding Box* ( $\text{fabs}(x - 0.5 * (x_{min} + x_{max}))$ ).
3. D'altra banda es calcula la meitat de l'amplada de la *Bounding Box* ( $x_{max} - x_{min}$ ) i se li suma una certa tolerància  $dist$ . Amb això s'obté la distància màxima al centre de la boundig box

$$(0.5 * (x_{max}-x_{min}) + dist).$$

4. Finalment, si la distància del canell al centre de la bounding box en valor absolut és més petita que una distància màxima al centre de la *Bounding Box*, s'acceptarà que el canell està agafant l'objecte en qüestió.
5. Cal repetir el mateix procés per la *y* amb el mateix procediment descrit.
6. Per tal d'assegurar-se que l'humà està agafant l'objecte, cal que es compleixin ambdues condicions, tant per les *x* com per les *y*.

#### 5.3.4 Funcions per mostrar el resultat

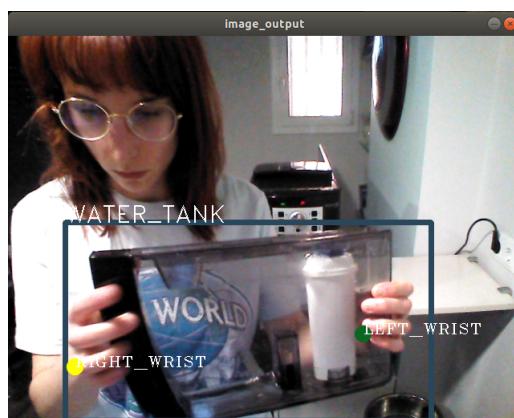
Per tal de veure si s'ha realitzat una bona detecció, cal mostrar el resultat en la imatge de sortida. És per això que es creen les següents funcions en la llibreria *CoffeeMachineROSNode* (vegeu [24]):

**plotBoundingBoxesInImage** - La funció *plotBoundingBoxesInImage*, els paràmetres d'entrada són la *x<sub>min</sub>*, la *y<sub>min</sub>*, la *x<sub>max</sub>*, la *y<sub>max</sub>* i una imatge d'entrada. Aquesta, permet mostrar en la imatge d'entrada un rectangle en les coordenades de la *Bounding Box* de la classe detectada, així com mostra el nom de la classe detectada.

**plotJointInImage** - La funció *plotJointInImage*, els paràmetres d'entrada són la *x*, la *y* i una imatge d'entrada. Aquesta, permet mostrar en la imatge d'entrada un cercle on es troben les coordenades del canell detectat, així com mostra el nom del canell.

#### 5.3.5 Resultat

El resultat tant de l'algoritme com de les funcions és mostra en la imatge 5.1. Aquesta mostra la localització dels dos canells, la localització d'un dipòsit d'aigua *water\_tank* i, l'aplicació de l'algoritme que permet detectar si l'humà està agafant l'objecte.



Imatge 5.1: Resultat de l'acció *Extreure el dipòsit d'aigua de la cafetera*

## Capítol 6

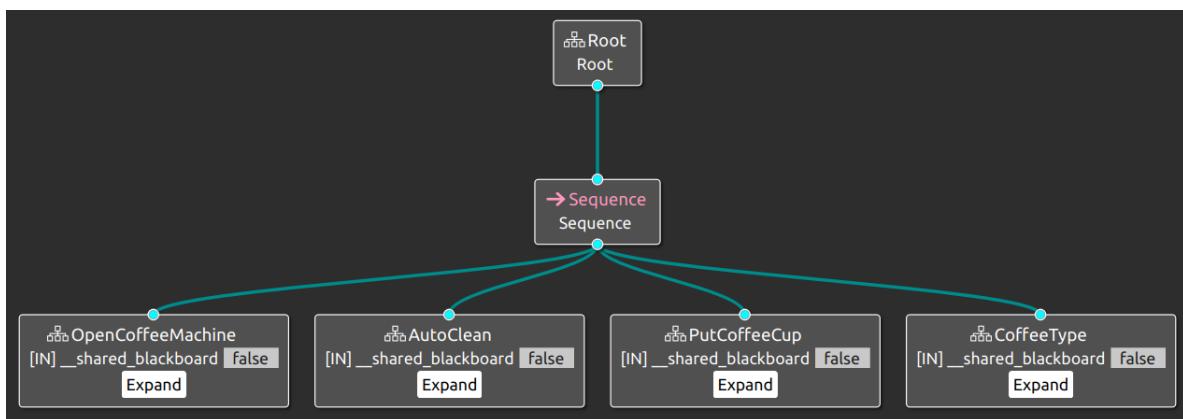
# DESENVOLUPAMENT DEL CONTROLADOR

### 6.1 Creació de l'arbre de comportament

El desenvolupament del controlador es basa en un arbre de comportament (vegeu l'apartat 3.2). Aquest serà el controlador base o intel·ligència que utilitzarà el robot per a detectar quines accions han estat acabades per l'humà i quines seran les següents en executar-se.

L'arbre a desenvolupar es basa en el diagrama de flux de la imatge 4.2. Amb això es procedirà a desenvolupar diferents subprocessos, accions, i condicions.

A la imatge 6.1 s'observa quatre subprocessos a desenvolupar, ordenades per ordre d'execució; Aquests els componen diferents accions i condicions.



Imatge 6.1: Distribució dels subprocessos amb l'eina *Groot*

#### 6.1.1 Accions

Les accions a desenvolupar són les que es mostren en la taula 6.1. El comportament respectiu, així com l'objecte clau a detectar es realitzarà tal com s'explica en l'apartat 5.

Acció	Descripció
AddMilkToCoffee	Afegir Llet
AddSugarToCoffee	Afegir Sucre
EmptyMarroTank	Buidar el dipòsit del marro
FillWaterTank	Omplir el dipòsit d'aigua
PlaceCleanCup	Afegir la tassa de neteja
PlaceCoffeeCup	Afegir la tassa de cafè
PressDesiredCoffee	Pressionar el cafè desitjat
SwitchOffCoffeeMachine	Apagar la cafetera
SwitchOnCoffeeMachine	Encendre la cafetera

Taula 6.1: Accions que formen part de l'arbre de comportament

### 6.1.2 Condicions

Les condicions a desenvolupar són les que es mostren en la taula 6.2. El comportament respectiu, així com l'objecte clau a detectar es realitzarà tal com s'explica en l'apartat 5.

### 6.1.3 Branques

Les branques a desenvolupar són les que es mostren en la taula 6.3 i es poden veure representades en la imatge 6.1.

#### Branca OpenCoffeeMachine

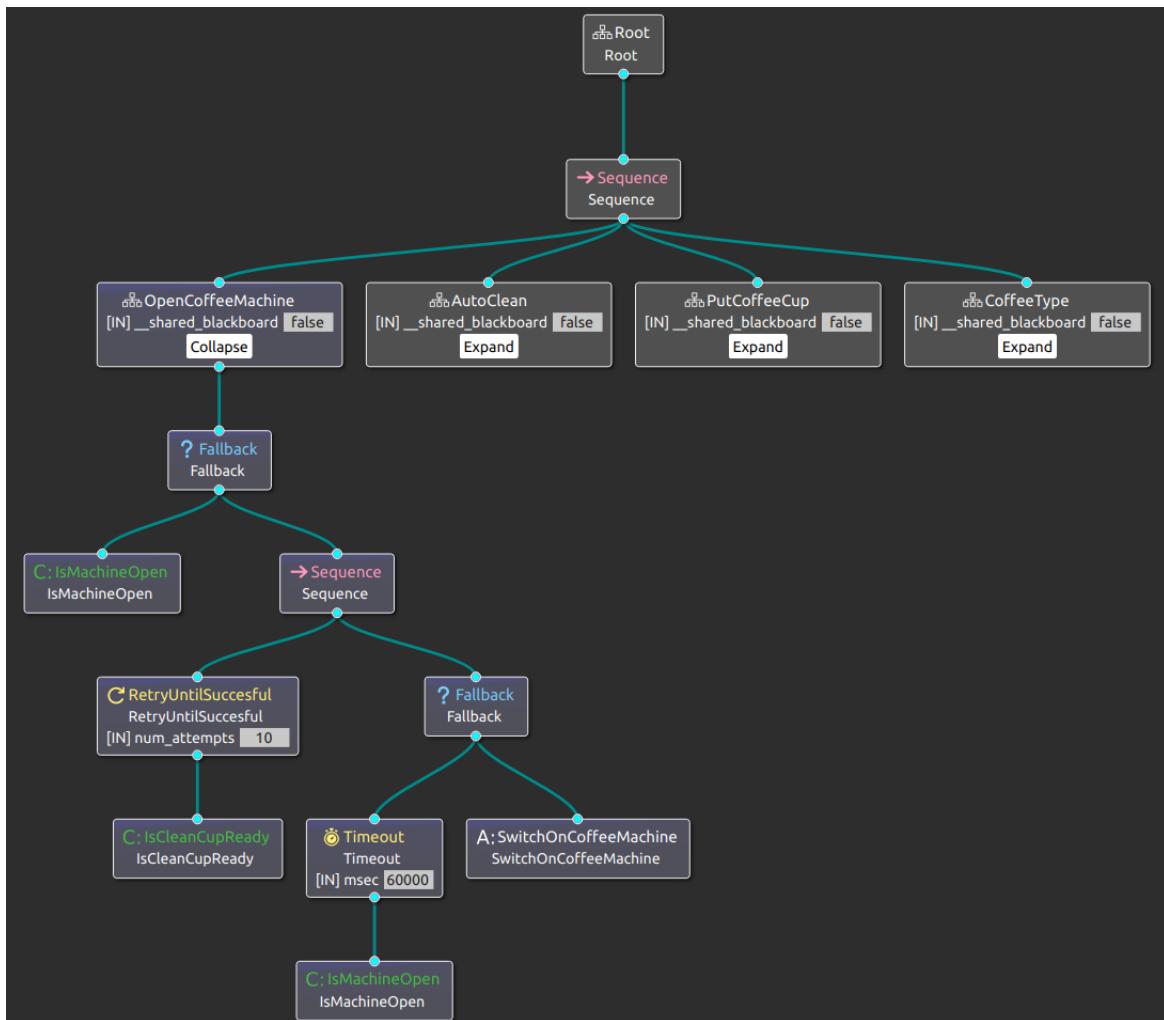
Aquesta branca es basa en el subprocés d'obrir la cafetera. Si s'observa la imatge 6.2, la branca consultarà la condició *IsMachineOpen*. Si és així, passarà a la següent branca *AutoClean*. Si no és així, comprovarà durant 10 intents la condició *IsCleanCupReady*, una vegada l'acció s'hagi efectuat satisfactòriament, perquè l'humà hi ha col·locat la tassa, comprovarà la condició *IsMachineOpen* durant 60 segons, per donar l'oportunitat a l'humà d'encendre la cafetera i si no, el robot procedirà a efectuar l'acció *SwitchOnCoffeeMachine*. Una vegada aquestes accions i condicions siguin satisfactòries, es procedirà amb la següent branca *AutoClean*.

Nom Condició	Condició a complir
<i>HasCupOfCoffeeBeenRemoved</i>	S'ha retirat la tassa de cafè de la cafetera?
<i>HasHumanAddedCleaningCup</i>	L'humà ha afegit una tassa de neteja?
<i>HasHumanAddedMilk</i>	L'humà ha afegit llet?
<i>HasHumanAddedSugar</i>	L'humà ha afegit sucre?
<i>IsCleanCupReady</i>	Està la tassa de neteja preparada en la cafetera?
<i>IsCleanProcessFinished</i>	Ha acabat el procés de neteja?
<i>IsCoffeeCupReady</i>	Està la tassa de cafè preparada en la cafetera?
<i>IsCoffeeFinished</i>	El cafè està llest?
<i>IsDesiredCoffeeSelected</i>	S'ha seleccionat el cafè desitjat?
<i>IsMachineOpen</i>	Està la cafetera oberta?
<i>IsMarroTankEmpty</i>	S'ha buidat el dipòsit de marro?
<i>IsMarroTankFull</i>	Està ple el dipòsit de marro?
<i>IsMarroTankPlacedInCoffeeMachine</i>	El dipòsit de marro està col·locat en la cafetera?
<i>IsMarroTankRemoved</i>	S'ha extret el dipòsit de marro?
<i>IsMilkDesired</i>	Es vol llet?
<i>IsSugarDesired</i>	Es vol sucre?
<i>IsThereEnoughWater</i>	Hi ha suficient aigua en la cafetera?
<i>IsWaterTankFull</i>	El dipòsit d'aigua està ple?
<i>IsWaterTankPlacedInCoffeeMachine</i>	El dipòsit d'aigua està col·locat en la cafetera?
<i>IsWaterTankRemoved</i>	El dipòsit d'aigua s'ha extret?

Taula 6.2: Condicions que formen part de l'arbre de comportament

Branca	Descripció
OpenCoffeeMachine	Subprocés d'obrir la cafetera
AutoClean	Subprocés de neteja
PutCoffeeCup	Subprocés d'afegir la tassa de cafè
CoffeeType	Subprocés per tal d'efectuar el cafè definitiu

Taula 6.3: Subprocessos que formen part de l'arbre de comportament

Imatge 6.2: Representació branca *OpenCoffeeMachine*

### Branca AutoClean

Aquesta branca es basa en el subprocés de neteja. Aquest el conforma tant l'operació d'omplir el dipòsit d'aigua en cas de carència com buidar el dipòsit de marro en cas de necessitat. Tal com s'observa a la imatge 6.3, la branca consultarà la condició *IsCleanProcessFinished*.

Si el resultat és un fracàs, seguirà amb el següent *fallback* per tal d'omplir el dipòsit d'aigua en cas de carència. En canvi, si el resultat és un èxit, seguirà amb la següent branca *PutCoffeCup*.

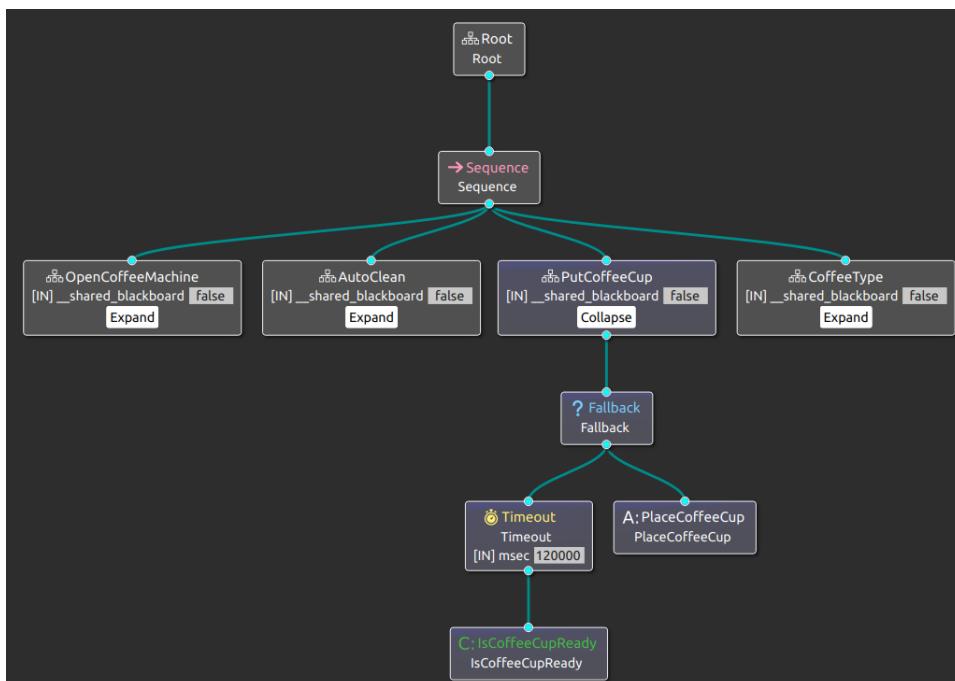
El següent *fallback* procedirà amb la comprovació de la condició *IsThereEnoughWater*. Si és un fracàs procedirà amb la seqüència encarregada d'omplir el dipòsit d'aigua i si no, procedirà amb el següent *fallback*, encarregat de buidar el dipòsit de marro. Aquesta seqüència primer comprovarà durant 120 segons la condició *IsWaterTankRemoved*, per donar temps a l'humà a enretirar-lo, quan retorni un èxit, comprovarà la condició *IsWaterTankFull* durant 120 segons per donar temps a l'humà a omplir-lo, si aquesta condició falla, el robot procedirà amb l'acció *FillWaterTank*. Una vegada aquesta acció sigui un èxit, procedirà a comprovar l'acció *IsWaterTankPlacedInCoffeMachine*

durant 120 segons esperant a que l'humà acabi de col·locar-lo bé i, una vegada aquesta acció retorni un èxit, procedirà a comprovar l'acció *IsCleanProcessFinished*. Si alguna d'aquestes condicions o accions d'aquesta seqüència falla procedirà amb l'últim *fallback*.

L'últim *fallback* procedirà a comprovar la condició invertida *IsMarroTankFull*, si és un fracàs procedirà a la seqüència encarregada de buidar el dipòsit de marro i si no, procedirà amb la següent branca *PutCoffeeCup*. La seqüència comprovarà la condició *IsMarroTankRemoved* durant 120 segons, una vegada aquesta condició sigui un èxit, comprovarà la condició *IsMarroTankEmpty* durant 120 segons, si aquesta retorna un fracàs, el mateix robot procedirà amb l'acció *EmptyMarroTank* i, si no, procedirà amb la següent condició *IsMarroTankPlacedInCoffeMachine* durant 15 intents, una vegada aquesta condició retorneu èxit procedirà amb la comprovació de l'acció *IsCleanProcessFinished*. Un cop completades les condicions/accions amb èxit, es procedeix amb la següent branca *PutCoffeeCup*.

### Branca PutCoffeeCup

Aquesta branca es basa en el subprocés d'afegir la tassa de cafè a la cafetera. Si s'observa la imatge 6.4 la branca comprovarà la condició *IsCoffeeCupReady* durant 120 segons, esperant que l'humà la posi si no hi ha cap tassa i vol fer ell aquesta tasca. Si aquesta falla, procedirà amb la següent acció *PlaceCoffeeCup*. Un cop col·locada la tassa, procedirà amb la següent branca *CoffeeType*.



imatge 6.4: Representació branca *PutCoffeeCup*

### Branca CoffeeType

Aquesta branca es basa en el subprocés de seleccionar el cafè desitjat, afegir llet i sucre en cas que hagi estat requerit, afegir la tassa de neteja i, apagar la cafetera en cas de necessitat. Si s'observa la imatge 6.5 aquesta branca conté un node de tipus seqüència.

En primer lloc consultarà la condició *IsDesiredCoffeSelected* durant 120 segons si una tasca la poden fer tant l'humà com el robot, el robot dóna l'oportunitat a l'humà de fer aquesta tasca, si aquesta falla, el robot procedirà a executar l'acció *PressDesiredCoffe*. Una vegada l'acció s'hagi efectuat amb èxit, es comprovarà la condició *IsCoffeFinished* durant 120 segons. Quan aquesta retorna èxit, la seqüència procedirà amb la següent condició, *HasCupOfCoffeBeenRemoved* la qual també es comprovarà durant 120 segons perquè pot ser que l'humà s'hi hagi avançat al robot i ell mateix hagi retirat la tassa. Una vegada aquesta condició també retorna èxit, durant 120 segons es comprovarà la condició *HasHumanAddedCleaningCup*, si aquesta falla, el robot procedirà a executar les accions *PlaceCleanCup* i *SwitchOffCoffeeMachine* de forma consecutiva, una rere l'altra. Si no falla, procedirà amb el següent *fallback*. Arribats a aquest punt, cal afegir llet i sucre al cafè (en cas que l'humà així vulgui el cafè) però, en tot cas, aquestes són tasques que es poden fer en paral·lel i les dues poden ser realitzades tant per l'humà com pel robot. D'aquesta manera el robot executa la tasca que l'humà no hagi escollit fer. Així que, durant 120 segons es comprovarà la condició *HasHumanAddedMilk* per donar-li temps a l'humà si posa la llet. Si aquesta falla, bé perquè l'humà no ha complert cap tasca encara o bé perquè ha escollit la tasca de posar sucre, el robot procedirà a comprovar la condició invertida *IsMilkDesired* si aquesta s'efectua amb èxit, vol dir que l'humà si vol llet per tant el robot procedirà a executar l'acció *AddMilkToCoffe*. Finalment,i de la mateixa forma que amb la llet, durant 120 segons es comprovarà la condició *HasHumanAddedSugar* per tal de donar-li temps a l'humà si vol posar-se sucre. Si aquesta falla el robot procedirà a comprovar la condició invertida *IsSugarDesired* si aquesta s'efectua amb èxit el robot procedirà a complir l'acció *AddSugarToCoffee*, on afegirà sucre al cafè.

### Forma d'emmagatzemar un arbre de comportament

L'aplicació *Groot* permet guardar l'arxiu creat en format \*.xml [8], fet que facilita l'accés i permet que pugui ser utilitzat per altres nodes. L'arxiu creat es pot trobar aquí [22].

## 6.2 Integració al paquet coffe\_machine

Per tal de poder treballar amb la mateixa arquitectura de ROS, cal integrar dins del paquet *coffe\_machine* aquest arbre de comportament. És aquí on s'utilitza la llibreria *behavior\_cpp\_v3* [9].

Si s'observa el codi *behavior\_tree\_node* del node *coffe\_machine* [21], permet efectuar els següents procediments:

- Conté mètodes que permeten enllaçar condicions simples (*registerSimpleCondition*) amb mètodes

de classes *CoffeMachineROSNode::IsMachineOpen* creades en el mateix node *coffee\_machine*. D'aquesta manera, s'indica en quina funció s'implementa el node condició de l'arbre del fitxer *.xml*.

```
factory . registerSimpleCondition (" IsMachineOpen " ,
std :: bind(&CoffeMachineROSNode :: IsMachineOpen , &coffe_machine_ros_node ));
```

- Conté mètodes que permeten enllaçar accions simples (*registerSimpleAction*) amb mètodes de classes *CoffeMachineROSNode::SwitchOnCoffeMachine* creades en el mateix node *coffee\_machine*. D'aquesta manera, s'indica en quina funció s'implementa el node acció de l'arbre del fitxer *.xml*.

```
factory . registerSimpleAction (" SwitchOnCoffeMachine " ,
std :: bind(&CoffeMachineROSNode :: SwitchOnCoffeMachine ,
&coffe_machine_ros_node ));
```

- Permet importar l'arxiu *\*.txt* vist en l'apartat 6.1.3 i crear un arbre de comportament. Cal prèviament tenir registrades totes les condicions i accions que hi apareguin.

```
if ( nh .getParam ( "/ coffee_machine / xml_path " , path )) {
    std :: cout << " xml path param ok " << std :: endl ;
}
auto tree = factory .createTreeFromFile ( path );
```

- Permet incorporar un *rati de velocitat*) per tal de demanar el resultat de l'acció o condició fins que retorna èxit o fallida i la integració amb l'arquitectura de ROS

```
while ( ros :: ok () && tree . tickRoot () == NodeStatus :: RUNNING )
{
    ros :: spinOnce ();
    rate . sleep (); // el temps que resta del cicle , dorm
}
```

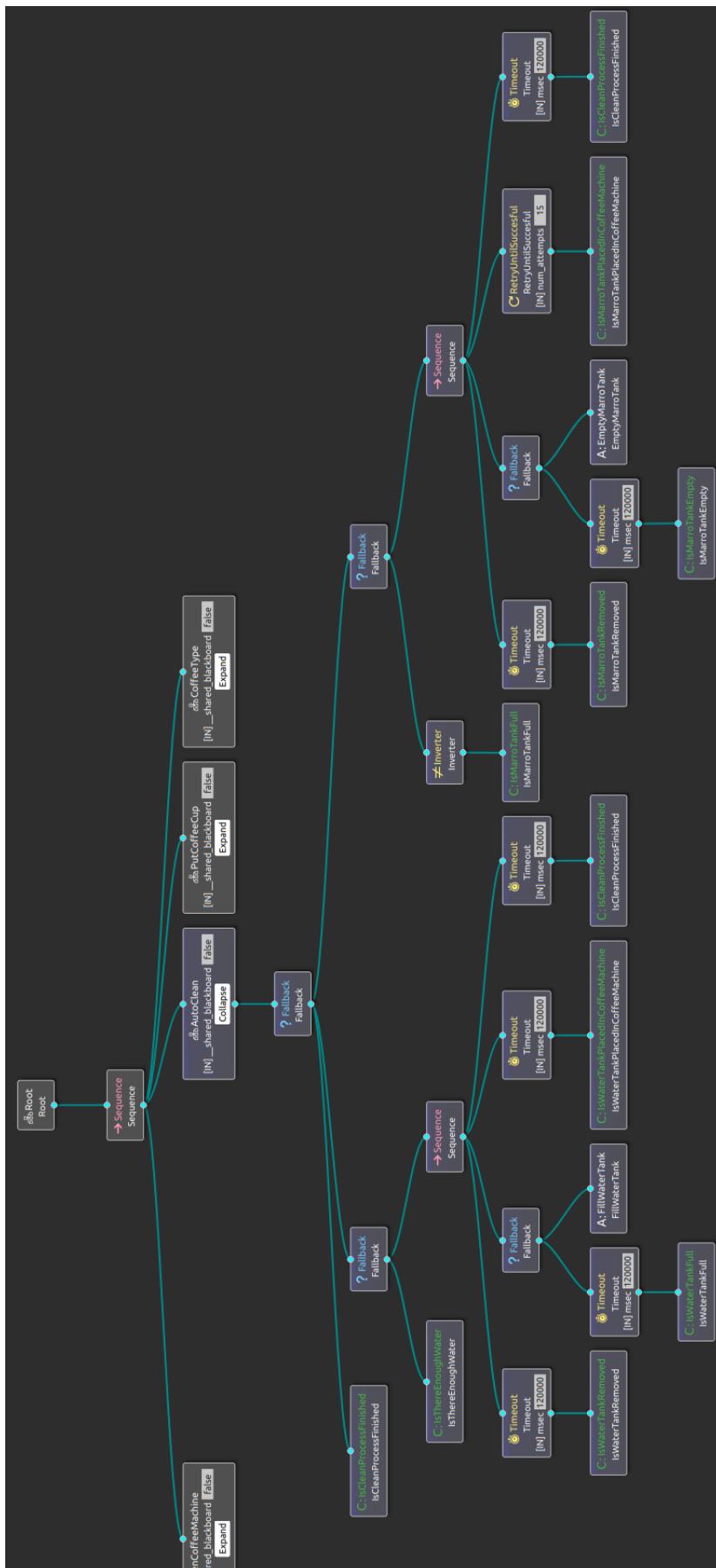
### 6.3 Creació del tòpic */coffee\_machine/Feedback*

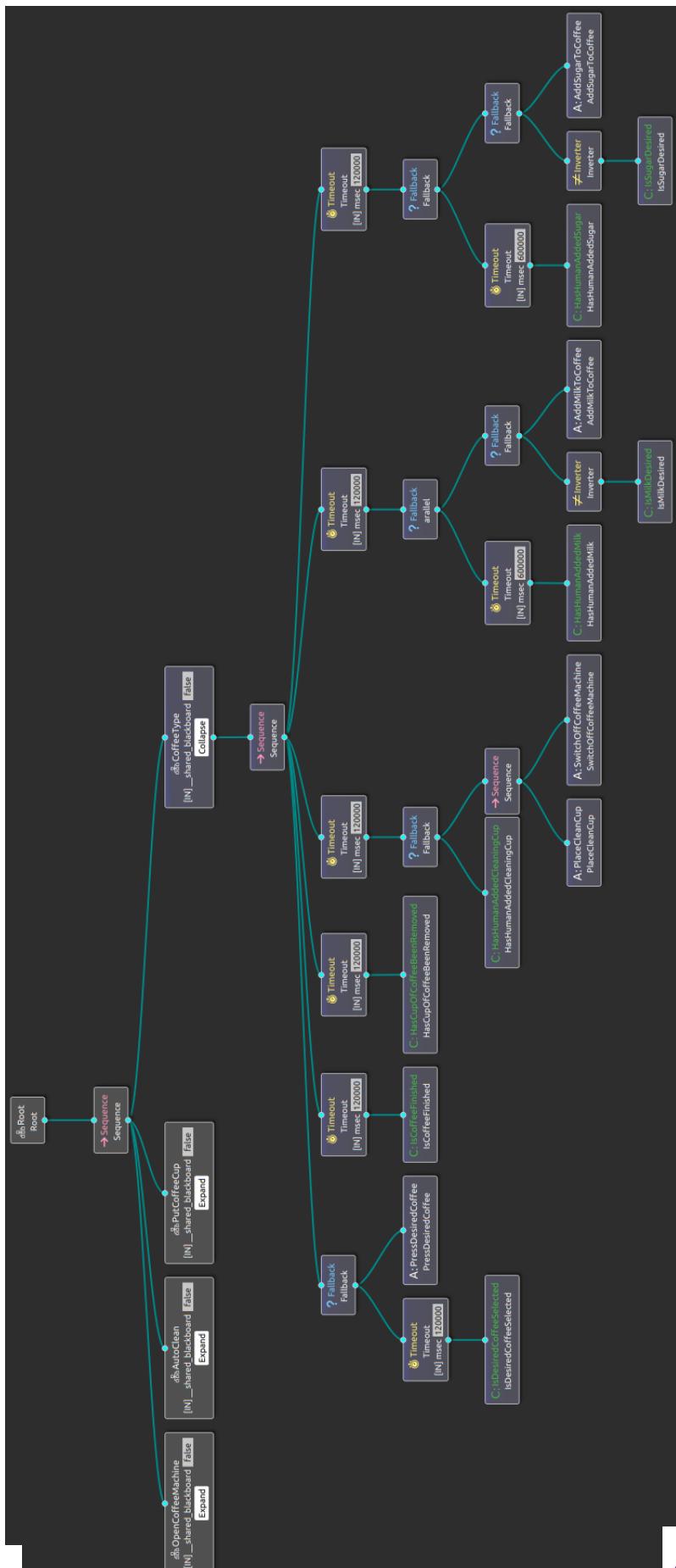
Per tal de tenir el control en tot moment de quines condicions o accions que s'estan executant en temps real en l'arbre de comportament, així com, saber en quin estat es troben, s'ha creat el tòpic */coffee\_machine/Feedback* (vegeu imatge 4.2) format pel missatge *rosmsg show coffee\_machine/State* (vegeu A.3).

Un exemple d'aquest tòpic és mostra en la imatge 6.6.

```
---  
NodeType: "HasCupOfCoffeBeenRemoved"  
NodeStatus: "RUNNING"  
---  
NodeType: "HasCupOfCoffeBeenRemoved"  
NodeStatus: "SUCCESS"  
---  
NodeType: "HasHumanAddedCleaningCup"  
NodeStatus: "FAILURE"  
---  
NodeType: "PlaceCleanCup"  
NodeStatus: "SUCCESS"  
---  
NodeType: "SwitchOffCoffeMachine"  
NodeStatus: "SUCCESS"  
---  
NodeType: "HasHumanAddedMilk"  
NodeStatus: "RUNNING"  
---  
NodeType: "HasHumanAddedMilk"  
NodeStatus: "RUNNING"  
---  
NodeType: "HasHumanAddedMilk"  
NodeStatus: "RUNNING"
```

imatge 6.6: Tòpic /coffe\_machine/Feedback





## Capítol 7

# VALIDACIÓ DEL CONTROLADOR I DELS EXPERIMENTS

Per tal de validar el controlador es generen els següents experiments:

1. **Primer experiment:** En aquest experiment es vol provar totes les branques dotant al robot de més prioritat, sobretot en l'acció d'obrir la cafetera, en la col·laboració humà-robot en omplir el dipòsit d'aigua, en la selecció del cafè desitjat i en l'acció d'apagar la cafetera. D'altra banda, l'humà haurà de reemplaçar la tassa de neteja, la de cafè i, ell mateix es posarà la llet i el sucre.
2. **Segon experiment:** En aquest experiment es vol provar totes les branques dotant a l'humà de més prioritat, sobretot a l'hora d'obrir la cafetera, col·locar la tassa de cafè, de neteja, de seleccionar el cafè desitjat, apagar la cafetera una vegada s'hagi realitzat el cafè i, finalment, es posarà la llet i el sucre. D'altra banda, el robot executarà les tasques col·laboratives d'omplir el dipòsit d'aigua i buidar el dipòsit de marro.
3. **Tercer experiment:** En aquest experiment es vol provar sobretot les branques *PutCoffeeCup* i *CoffeeType*. És per això que es parteix d'entrada on la cafetera ja està oberta i, que el procés de neteja ja està acabat. Tanmateix, es vol donar la màxima prioritat al robot en les tasques d'afegir llet i sucre i, l'humà, seleccionarà el cafè i col·locarà la tassa de neteja.

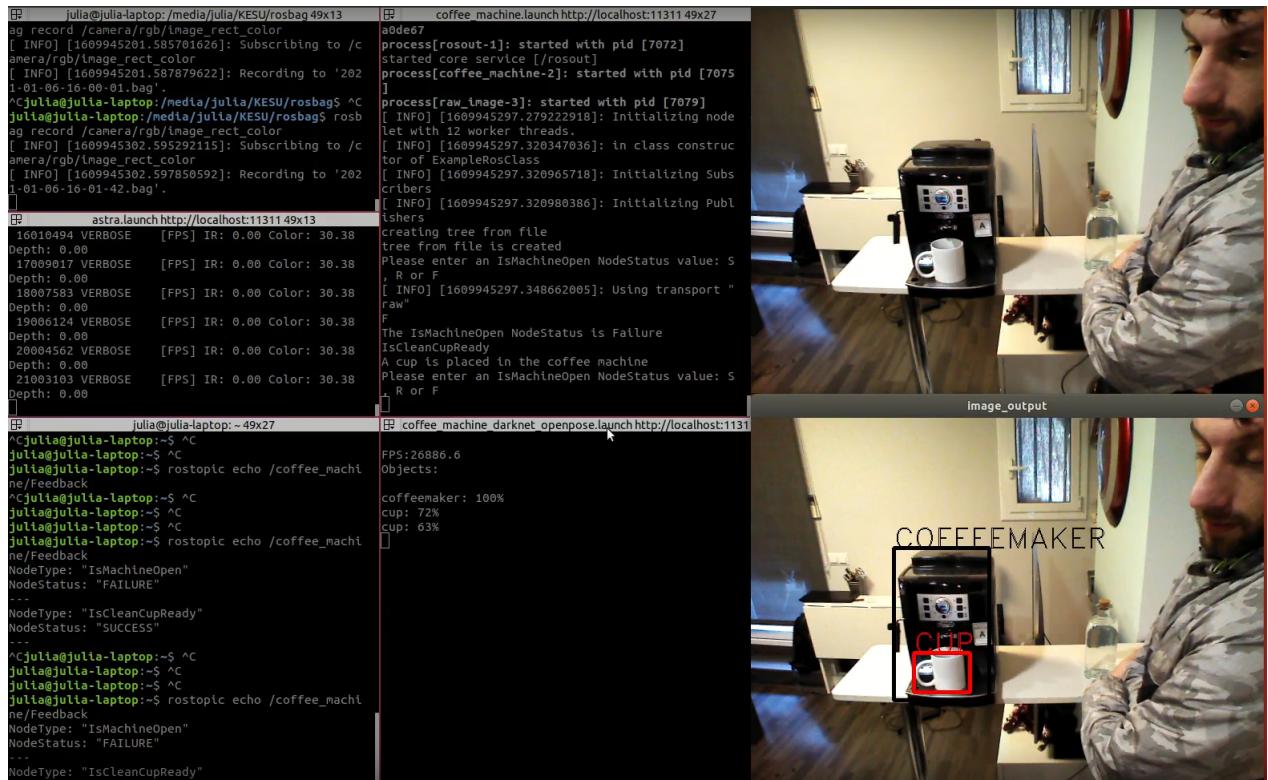
Aquests es poden trobar dins de la carpeta [25].

### 7.0.1 Primer Experiment:

El primer experiment conté accions i condicions distribuïdes segons les branques vistes a l'apartat 6.1.3.

En primer lloc es comença amb la branca **OpenCoffeeMachine**; El fet que la condició *IsMachineOpen* sigui *Failure*, imposat per aquest experiment, fa que sigui necessari procedir amb la

condició *IsCleanCupReady*, la qual és capaç de detectar la tassa, tal com és mostra en la imatge 7.1. Seguidament, la condició *IsCleanCupReady* es força a ser *Failure* i, per tant, el robot procedeix a executar l'acció *SwitchOnCoffeeMachine*.



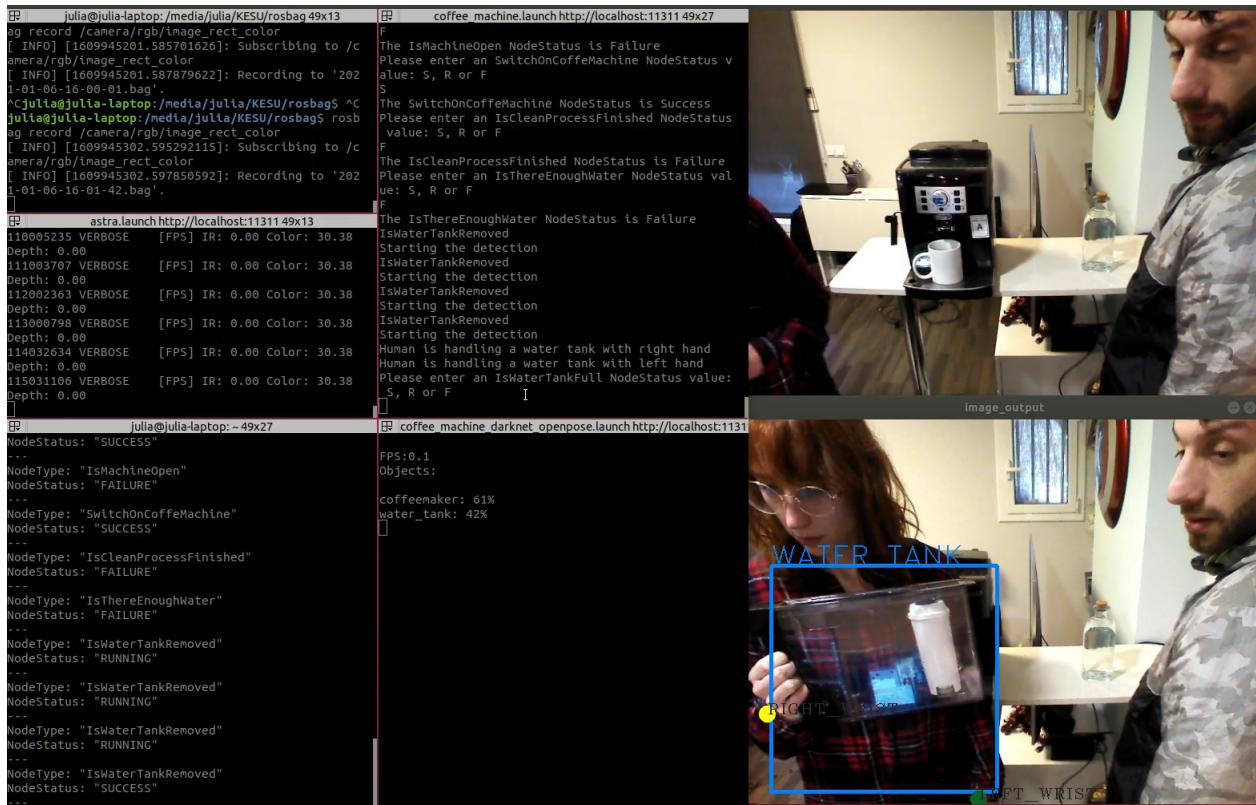
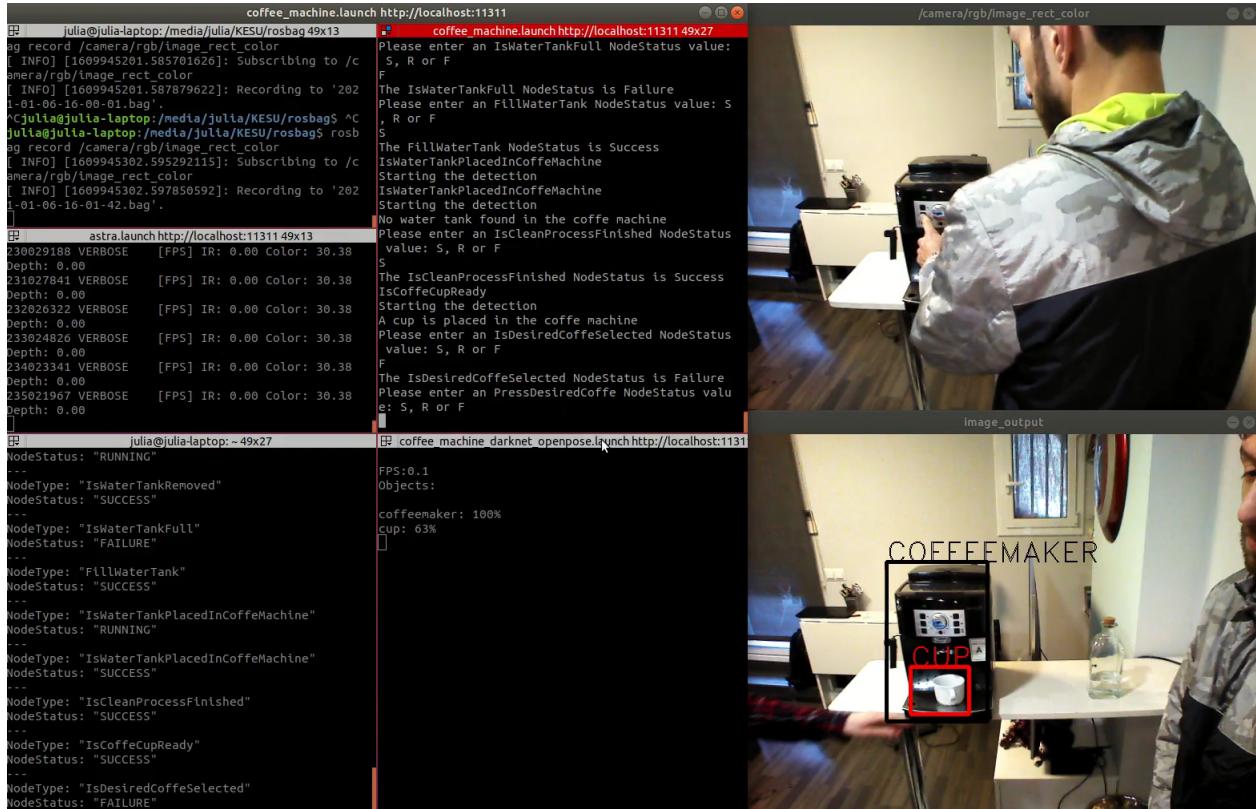
Imatge 7.1: Resultat condició *IsCleanCupReady*

En segon lloc, es procedeix amb la branca **AutoClean**; Com que les condicions *IsCleanProcess-Finished* i *IsThereEnoughWater* són *failure* es procedeix amb la condició *IsWaterTankRemoved*, on es detecta a l'humà agafant el dipòsit d'aigua, tal com és mostra en la imatge 7.2.

Seguidament, i, com que la condició *IsWaterTankFull* és *failure*, l'humà procedeix a executar l'acció *FillWaterTank* conjuntament amb el robot. Una vegada l'acció està acabada satisfactòriament, s'executa la condició *IsWaterTankPlacedInCoffeMachine*, la qual també acaba amb èxit.

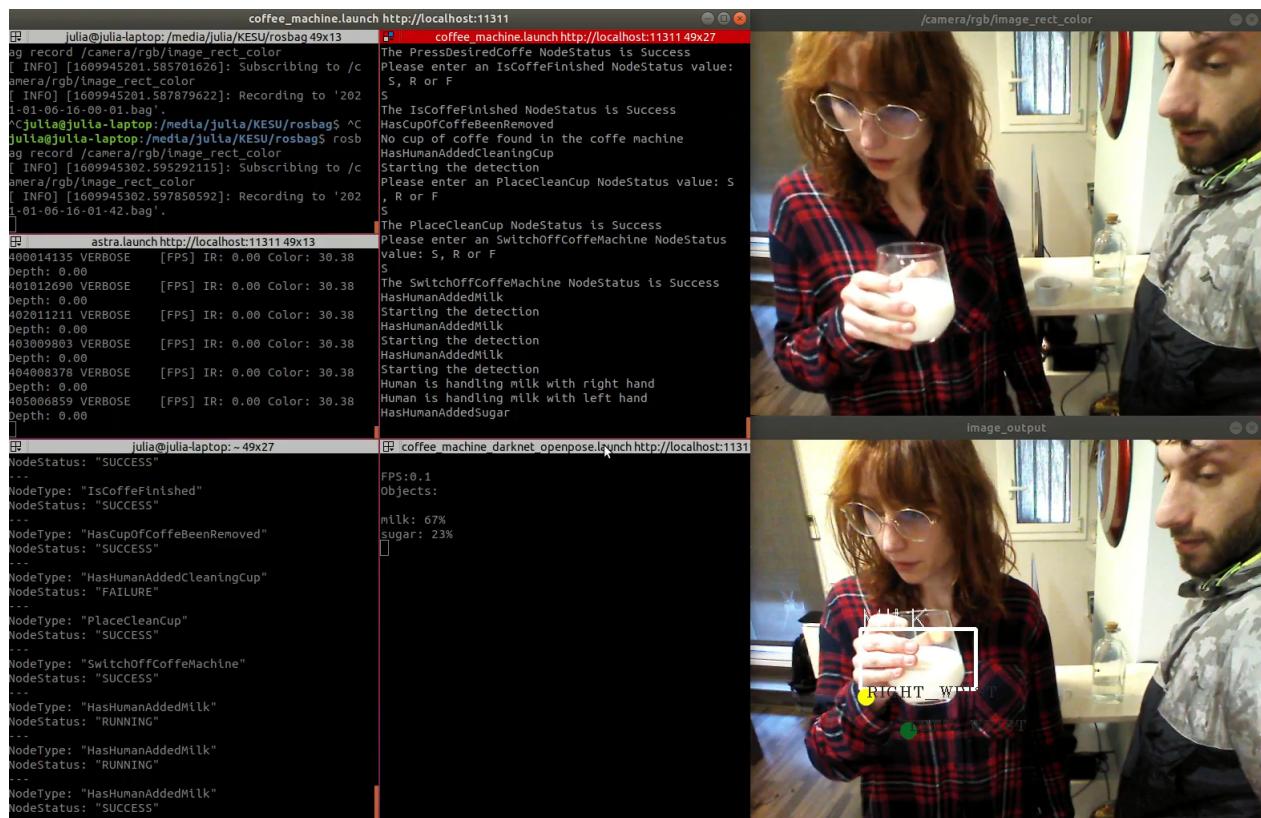
Una vegada ha acabat el procés de neteja satisfactòriament, es força la condició *IsCleanProcessFinished* a èxit. Per tant, l'algoritme passa a executar la següent branca **IsCoffeCupReady**.

En tercer lloc, procedeix amb la branca **PutCoffeeCup**; amb la primera condició *IsCoffeCupReady*, tal com es mostra en la imatge 7.3 com que aquesta s'efectua amb èxit, això permet passar a la següent branca *CoffeeType*.

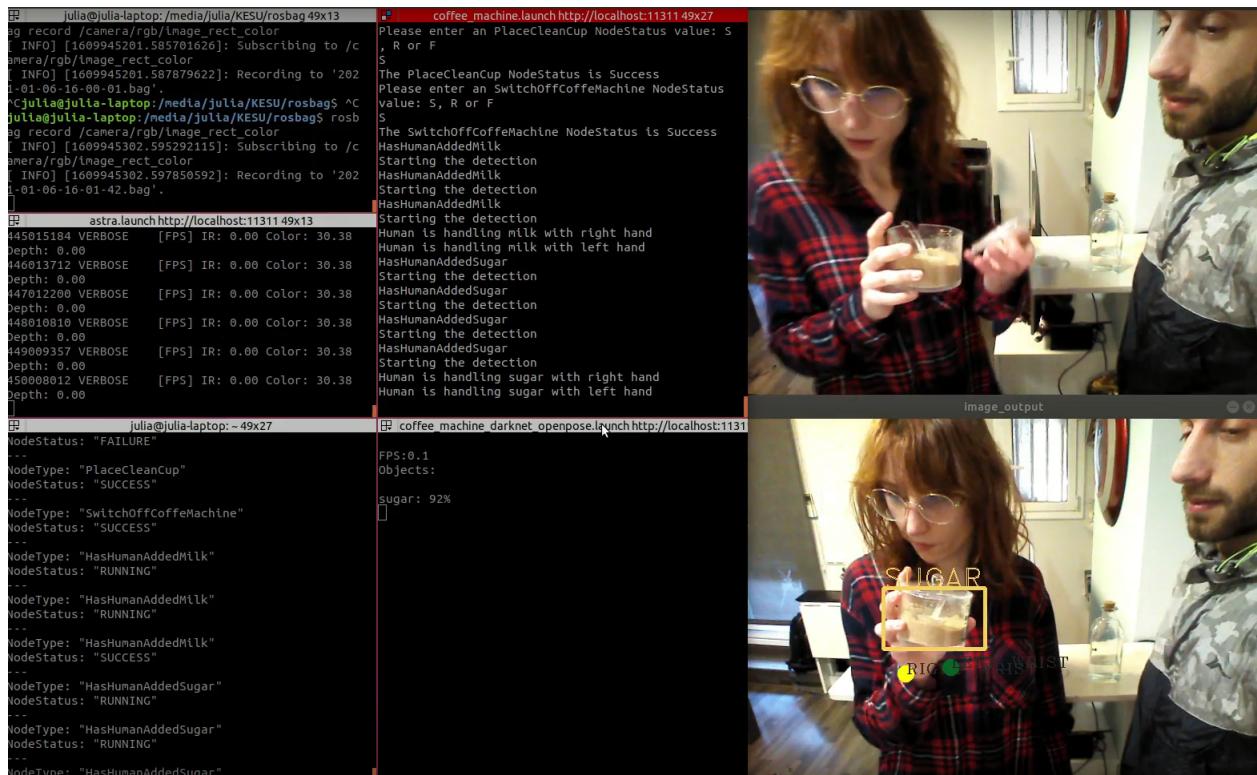
Imatge 7.2: Resultat condició *Is WaterTankRemoved*Imatge 7.3: Resultat condició *IsCoffeCupReady*

En quart lloc i últim, es procedeix amb la branca **CoffeeType**; amb la condició *IsDesiredCoffeeSelected*. Com que aquesta condició és força perquè falli, l'algoritme passa a la següent acció, *PressDesiredCoffe*, la qual és executada pel robot. Seguidament, una vegada la condició *IsCoffeFinished* és executada amb èxit es continua amb la condició *HasCupOfCoffeBeenRemoved*, la qual no detecta cafè per enllot, fet que procedeix a executar la condició *HasHumanAddedCleaningCup*. Com que aquesta condició no troba la tassa de neteja, l'humà procedeix a executar la tasca *PlaceCleanCup* de forma satisfactòria i, el robot procedeix a complir l'acció *SwitchOffCoffeMachine*.

Una vegada la cafetera està apagada, es procedeix a efectuar condició *HasHumanAddedMilk*, on l'humà conté un got de llet a la mà, tal com és mostra en la imatge 7.4, fet que indica al robot que l'humà ha optat per executar la tasca de posar llet. Seguidament, continua amb la condició *HasHumanAddedSugar* com que aquesta retorna *success*, ja que l'humà té un pot de sucre a la mà, tal com és mostra en la imatge 7.5. Per tant es dóna per finalitzat el primer experiment.



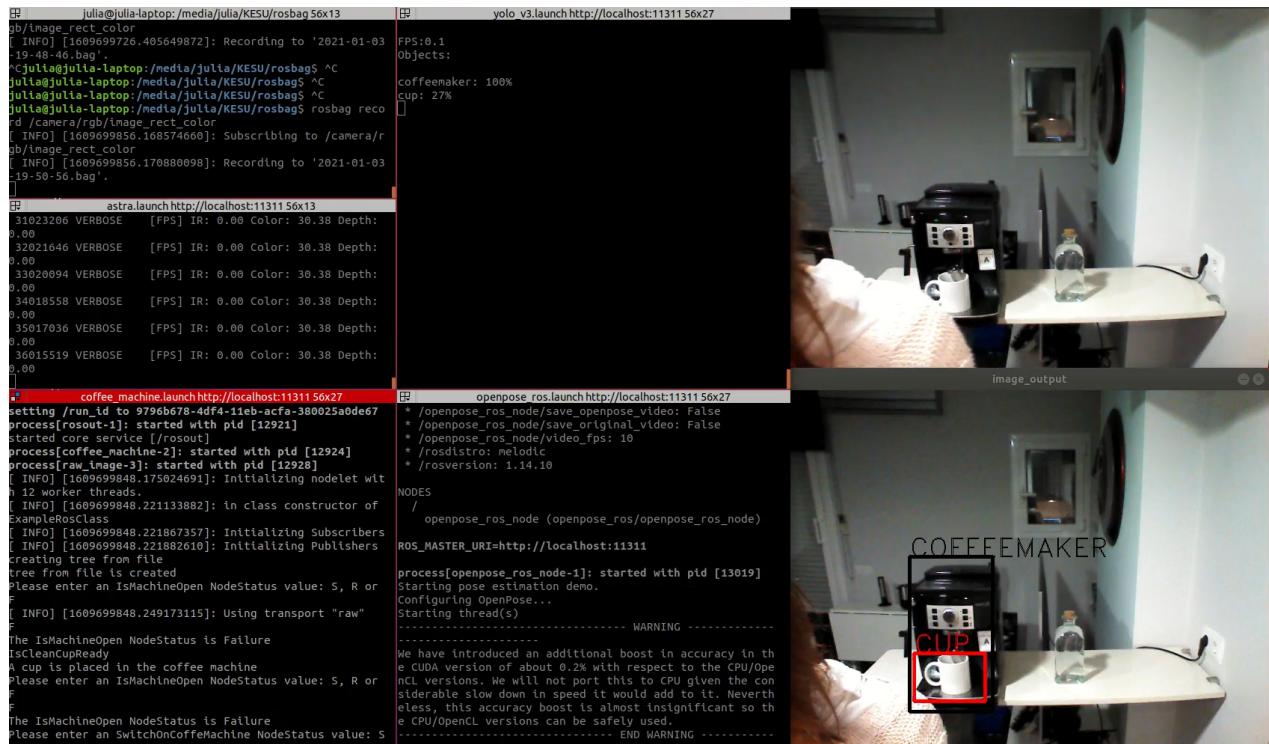
Imatge 7.4: Resultat condició *HasHumanAddedMilk*

Imatge 7.5: Resultat condició *HasHumanAddedMilk*

## 7.1 Segon Experiment:

El segon experiment conté accions i condicions distribuïdes segons les branques vistes a l'apartat 6.1.3.

Primerament es comença amb la branca **OpenCoffeeMachine**; El fet que la condició *IsMachineOpen* sigui *Failure*, imposat per aquest experiment, fa que sigui necessari procedir amb la condició *IsCleanCupReady*, la qual és capaç de detectar la tassa, tal com és mostra en la imatge 7.6. Seguidament, la condició *IsCleanCupReady* és força a ser *Failure* i, per tant, el robot procedeix amb l'acció *SwitchOnCoffeeMachine*.



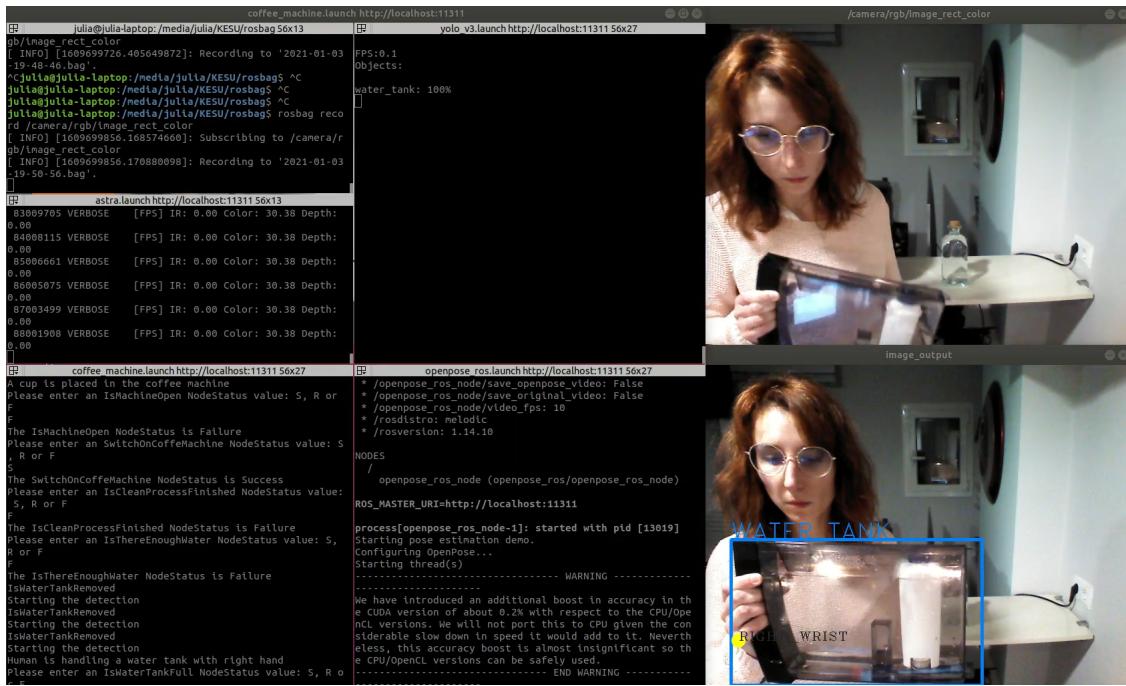
Imatge 7.6: Resultat condició *IsCleanCupReady*

En segon lloc, es procedeix amb la branca **AutoClean**; Com que les condicions *IsCleanProcessFinished* i *IsThereEnoughWater* són *failure* es procedeix amb la condició *IsWaterTankRemoved*, on es detecta a l'humà agafant el dipòsit d'aigua, tal com es mostra en la imatge 7.7.

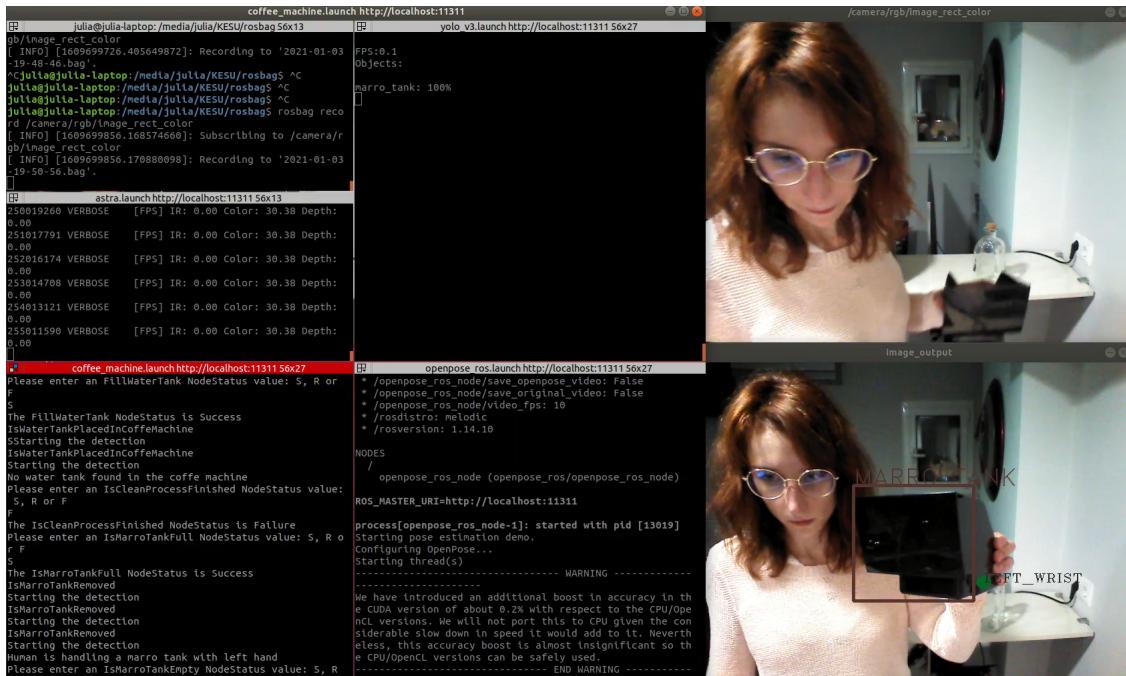
Seguidament, i, com que la condició *IsWaterTankFull* és *failure*, l'humà procedeix a executar l'acció *FillWaterTank* conjuntament amb el robot. Una vegada l'acció està acabada satisfactòriament, s'executa la condició *IsWaterTankPlacedInCoffeMachine*, la qual també acaba amb èxit.

Seguidament, la condició *IsCleanProcessFinished* és *failure*, per tant es procedeix amb l'acció *IsMarroTankFull*, el qual és capaç de detectar a l'humà agafant el dipòsit de marro, tal com és mostra en la imatge 7.8. Seguidament, es procedeix amb la condició *IsMarroTankEmpty*, com que aques-

ta condició és *failure* es procedeix amb l'acció EmptyMarroTank on es realitza una col·laboració humà-robot de forma satisfactoria. Tanmateix, s'executa la condició *IsMarroTankPlacedIncoffeeMachine*, la qual també acaba amb l'èxit de la no detecció del dipòsit de marro, ja que l'humà ja l'ha col·locat

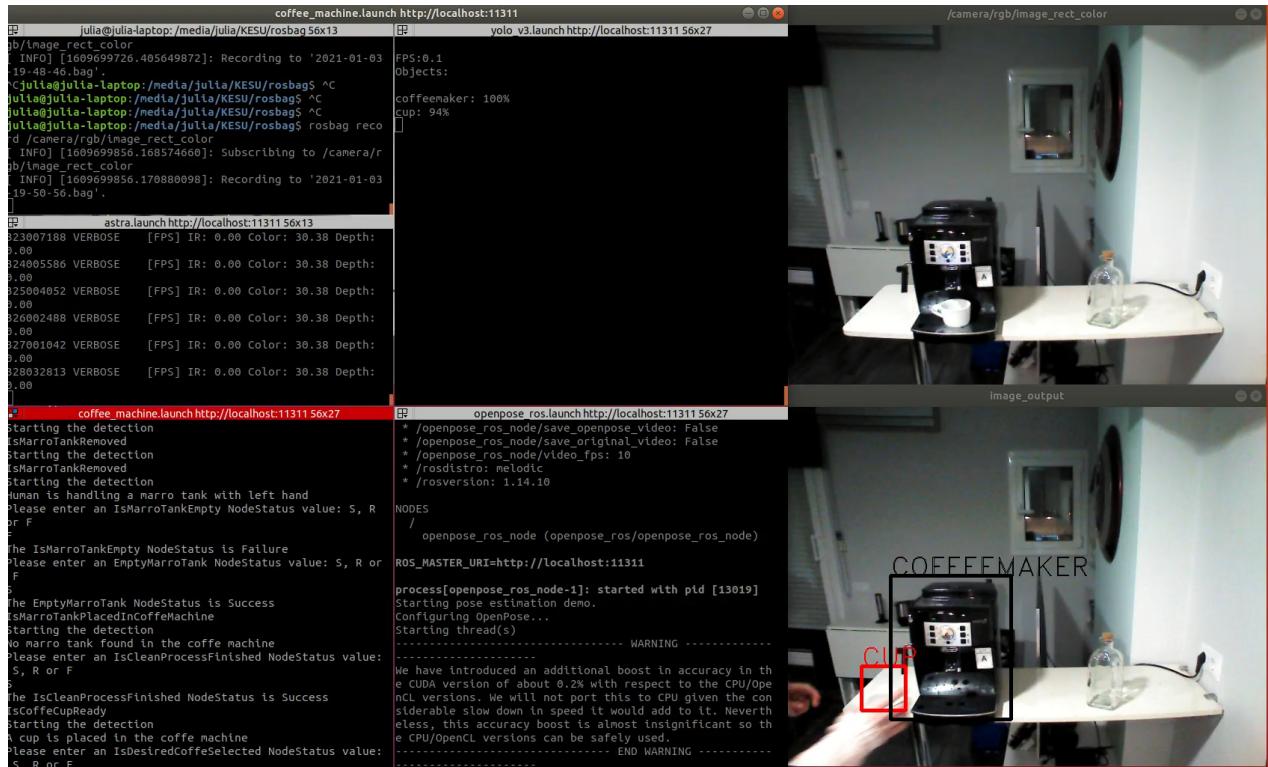


Imatge 7.7: Resultat condició *IsWaterTankRemoved*



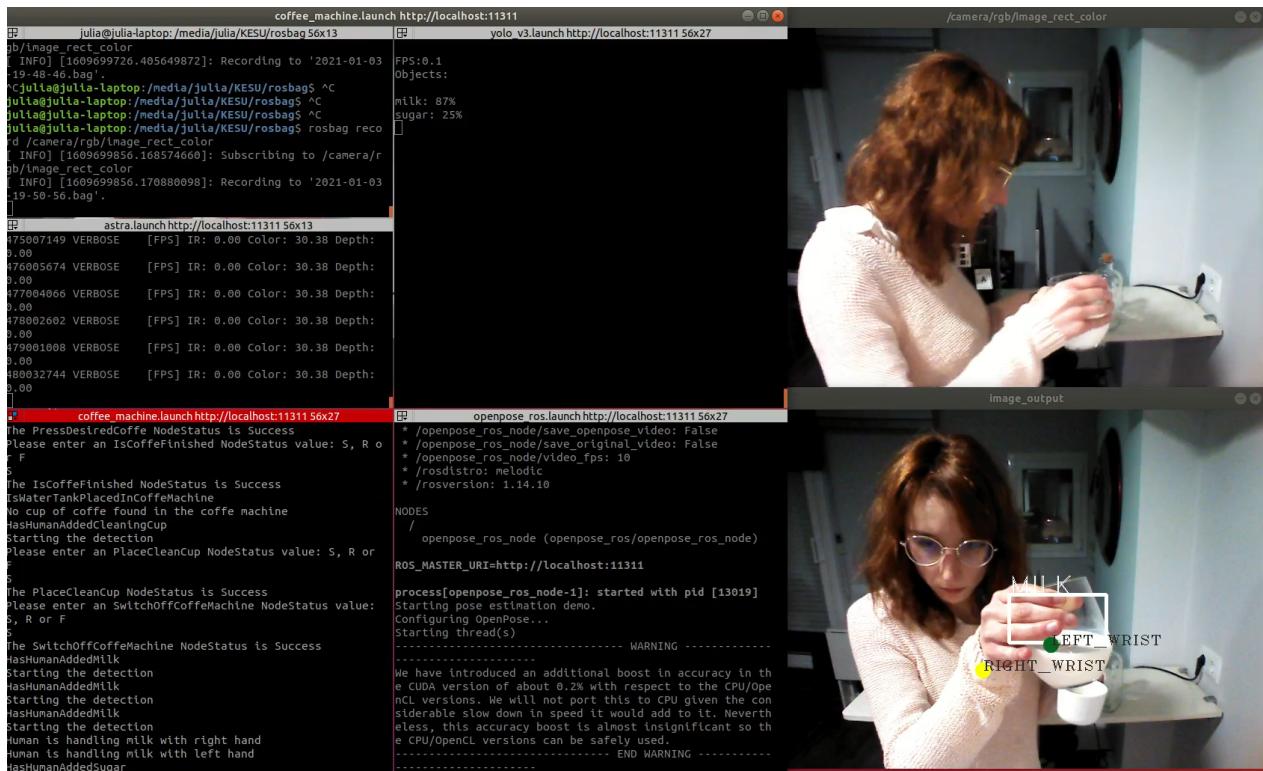
Imatge 7.8: Resultat condició *IsMarroTankRemoved*

En tercer lloc, procedeix amb la branca **PutCoffeeCup**; amb la primera condició *IsCoffeCupReady*, tal com es mostra en la imatge 7.9 com que aquesta s'efectua amb èxit, això permet passar a la següent branca *CoffeeType*.



Imatge 7.9: Resultat condició *IsCoffeCupReady*

Finalment, es procedeix amb la branca **CoffeeType**; amb la condició *IsDesiredCoffeSelected*. Com que el resultat d'aquesta retorna *failure*, procedeix a executar l'acció *PressDesiredCoffe*, la qual és desenvolupada per l'humà amb èxit. Seguidament, una vegada la condició *IsCoffeFinished* és executada amb èxit es continua amb la condició *HasCupOfCoffeBeenRemoved*, la qual no detecta cafè per enllac, fet que procedeix a executar la condició *HasHumanAddedCleaningCup*. Com que aquesta condició no troba la tassa de neteja, el robot entén que l'humà no ha dut a terme aquesta tasca i per tant l'ha de fer ell realitzant l'acció *PlaceCleanCup* de forma satisfactòria i, el robot procedeix a complir l'acció *SwitchOffCoffeMachine*. Una vegada la cafetera està apagada, es procedeix a efectuar condició *HasHumanAddedMilk*, on l'humà conté un got de llet a la mà, tal com es mostra en la imatge 7.10, fet que indica al robot que l'humà ha optat per executar la tasca de posar llet. Seguidament, continua amb la condició *HasHumanAddedSugar* com que aquesta retorna *failure*, s'executa la condició *IsSugarDesred* la qual s'executa amb èxit. És per això que finalment s'efectua l'acció *AddSugarToCoffe*, la qual és executada pel robot paral·lelament mentre l'humà hi posi llet.

Imatge 7.10: Resultat condició *HasHumanAddedMilk*

## 7.2 Tercer Experiment:

El tercer experiment conté accions i condicions distribuïdes segons les branques vistes a l'apartat 6.1.3. Aquest experiment no conté el procés de neteja.

Primerament es comença amb la branca **OpenCoffeeMachine**; El fet que la condició *IsMachineOpen* sigui *Success*, imposat per aquest experiment, fa que sigui necessari procedir amb la condició *IsCleanCupReady*, la qual és capaç de detectar la tassa. Seguidament, la condició *IsCleanCupReady* és força a ser *Failure* i, per tant, el robot procedeix amb l'acció *SwitchOnCoffeeMachine*.

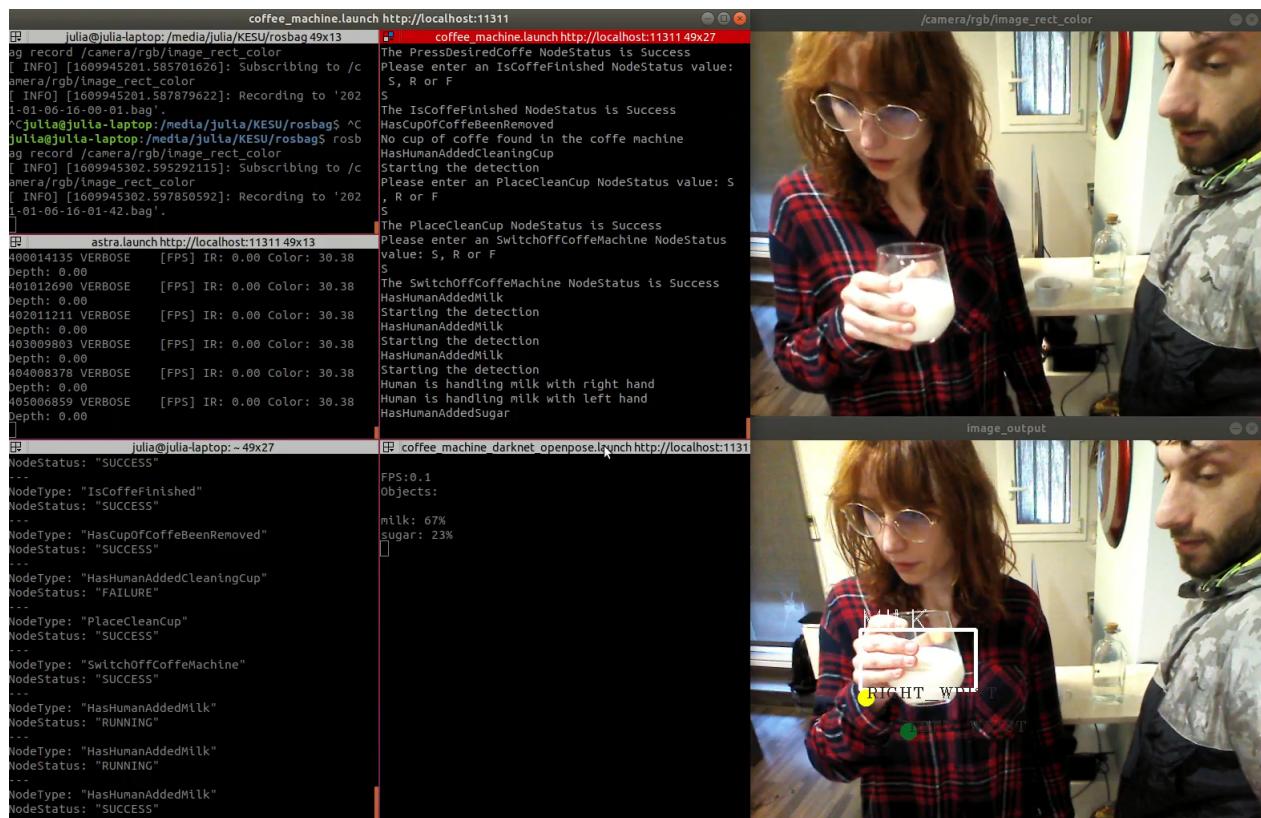
En segon lloc, es procedeix amb la branca **AutoClean**; Com que la condició *IsCleanProcessFinished* és imposta com a *Success*, passa a executar la següent branca *PutCoffeeCup*,

En tercer lloc, procedeix amb la branca **PutCoffeeCup**; amb la primera condició *IsCoffeeCupReady*, on, es detecta la tassa conjuntament amb la cafetera.

Finalment, es procedeix amb la branca **CoffeeType**; amb la condició *IsDesiredCoffeSelected*. Com que el resultat d'aquesta retorna *success*. Procedeix a executar la següent condició *IsCoffeeFinished* és executada amb èxit es continua amb la condició *HasCupOfCoffeBeenRemoved*, la qual no detecta cafè per enllot, fet que procedeix a executar la condició *HasHumanAddedCleaningCup*. Com que aquesta condició no troba la tassa de neteja, l'humà realitza l'acció *PlaceCleanCup*, on col-loca la tassa a la cafetera de forma satisfactòria i, el robot procedeix a complir l'acció *SwitchOffCoffeMachine*. Una vegada la cafetera està apagada, es procedeix a efectuar condició

*HasHumanAddedMilk*, com que al cap d'un temps l'acció de posar llet no és efectuada per l'humà, l'arbre de comportaments executa la condició *IsMilkDesired*, com que aquesta és força a *success*, el robot procedeix a posar la llet al cafè, realitzant l'acció *AddMilkToCoffee*. Seguidament, com que al cap d'un temps l'acció de posar sucre *HasHumanAddedSugar* no és efectuada per l'humà, l'arbre de comportaments executa la condició *IsSugarDesred*, com que aquesta és força a *success*, el robot procedeix a posar la llet al cafè realitzant l'acció *AddSugarToCoffee*.

on l'humà conté un got de llet a la mà, tal com és mostra en la imatge 7.11, fet que indica al robot que l'humà ha optat per executar la tasca de posar llet. Seguidament, continua amb la condició *HasHumanAddedSugar* com que aquesta retorna *failure*, s'executa la condició *IsSugarDesred* la qual s'executa amb èxit. És per això que finalment s'efectua l'acció *AddSugarToCoffee*, la qual és executada pel robot paral·lelament a que l'humà hi posi llet.



Imatge 7.11: Resultat condició *HasHumanAddedMilk*

## Capítol 8

# Anàlisi econòmic

En aquest capítol es detalla el pressupost associat al desenvolupament de l'aplicació, les llicències i, el hardware necessari. Com que el projecte es basa en la implementació d'una nova aplicació prototíp, no s'ha efectuat un estudi de la viabilitat Econòmica de l'aplicació.

### 8.1 Costos

Durant els processos de disseny, construcció i desenvolupament descrits a la memòria, es va fer un seguiment de les despeses. Els costos associats es classifiquen en costos dels materials utilitzats i costos d'enginyeria, els quals es troben detallats en les taules 8.1 i 8.2, respectivament. El cost de l'hora d'enginyer s'ha establert en 25 €/hora.

Taula amb els costos del material és la següent:

Component	Cost(€)
Ordenador ASUS ROG STRIX G [29]	1066€
Quota google colab pro [5]	25.95€
Astra camera ( <i>amortitzada</i> )	0 €
<b>TOTAL</b>	<b>1091.95</b>

Taula 8.1: Costos del Material

Cal especificar que les hores de desenvolupament de l'aplicació s'han aproximat a uns 60 dies, treballant unes 8h/dia. Sumant el cost total de les dues taules 8.1 i 8.2 s'obté que el cost del projecte és de **13092 [€]**.

Concepte	Hores	preu/hora	Cost(€)
Desenvolupament de l'aplicació	25	480	12000
<b>TOTAL</b>			<b>12000</b>

Taula 8.2: Costos d'Enginyeria

## Capítol 9

# Impacte mediambiental

En aquest capítol es fa un estudi de l'impacte mediambiental associat a la realització del projecte. S'han de distingir tres etapes diferents en les quals hi ha la possibilitat d'emetre residus al medi ambient: el procés de desenvolupament, la vida útil de l'aplicació amb els respectius elements utilitzats, i el final de la vida útil d'aquests. Per tant, s'ha dividit l'estudi en aquestes tres etapes:

### 9.1 Desenvolupament:

En aquesta etapa es pot distingir els components adquirits i utilitzats mena de modificació, com per exemple l'ordinador i la càmera així com l'impacte associat a l'energia elèctrica consumida per entrenar el model.

### 9.2 Vida útil

La vida útil de l'ordinador es basa en 7 anys si el deteriorament és causat per la seva utilització. Si es fa un mal ús de l'ordinador, aquest es farà malbé abans, per tant, cal tenir present que per utilitzar l'ordinador es necessita energia elèctrica. A causa del fet que la producció d'aquesta energia provoca emissions al medi ambient cal fer un ús responsable d'aquests.

### 9.3 Final de la vida útil

Un cop arribat el final de la vida útil de l'ordinador i la càmera, aquests han de ser desmuntat i reciclat. La major part de la seva estructura és de plàstic, el qual s'ha de dipositar en el contenidor pertinent. A més a més, contenen cargols i parts metàl·liques d'acer, així que tots els components d'aquests materials són fàcilment desmuntables, classificables, reciclables i es poden portar a forns. Tot el cablejat de l'ordinador s'ha de portar a la deixalleria, per tal que es pugui separar correctament el coure del recobriment de plàstic. Tanmateix, les bateries de liti s'han de dipositar en el contenidor pertinent, ja que són un material molt contaminant.

## Capítol 10

# Conclusions

L'objectiu general del projecte era la implementació d'un mòdul de seguiment de les accions d'un humà i dels seus gestos en la realització d'un cafè, per tal d'adaptar el comportament del robot.

En primer lloc, s'ha efectuat un estudi previ i validació de les eines necessàries per al correcte desenvolupament de l'aplicació, així com la respectiva integració amb ROS. Seguidament, es defineix quina és l'arquitectura a desenvolupar basada en l'estructura de l'aplicació.

En segon lloc, per tal de detectar els objectes desitjats es procedeix amb l'entrenament de la xarxa neuronal de les 7 classes que conformen l'escenari de treball. D'aquí s'obté un resultat amb una molt bona precisió. Seguidament, es procedeix a la integració amb ROS, facilitant la respectiva consolidació amb l'aplicació.

En tercer lloc, per tal de ser capaços de detectar gestos humans o, accions, és necessari la detecció de l'humà. És per això que s'integra el mòdul de detecció de gestos amb ROS i conseqüentment s'afegeix la detecció dels obstacles prèviament creada.

Finalment, es crea l'arbre de comportament del procés a executar conjuntament amb la detecció d'accions prèviament creades. Això conforma el paquet que conté el mòdul cognitiu.

Per tal de provar si la llum és un factor important en la detecció, s'han realitzat tres proves en el mateix escenari tipus, fent referència a diferents moments del dia. Els resultats són els següents:

En la primera prova tots els objectes i les articulacions són detectades i, l'arbre de comportament s'executa de forma correcta. S'observa una lleugera desviació en la detecció del canell esquerre. Aquesta dificultat en la detecció del braç esquerre està ocasionada per la falta de llum en el braç esquerre.

En la segona prova es detecten tots els objectes desitjats a excepció del sucre, això pot ser degut al fet que la prova es va efectuar de forma nocturna. La falta de llum en l'entorn dificulta la detecció dels objectes. Malgrat això, com que la noia porta una samarreta més clara, això facilita a l'algoritme de detecció d'articulacions la detecció.

En la tercera prova, és capaç de detectar tots els objectes desitjats i el comportament de

l'algoritme és l'òptim.

És per això que es dóna per vàlid el disseny de l'algoritme implementat, amb alguns aspectes a millorar.

Des d'un punt de vista econòmic i d'impacte ambiental, no es considera cap inconvenient a la viabilitat de la implementació de l'algoritme cognitiu.

Tot i així, cal considerar que aquestes anàlisis s'han fet basant-se en els recursos utilitzats en el present projecte, i és necessari que els projectes que es realitzin a posteriori, facin unes anàlisis específic i exhaustiu de les repercussions ambientals i econòmiques.

Amb tot això, s'ha aconseguit establir una base i una guia a partir de la qual es pugui seguir treballant en la detecció d'accions humanes, i per poder crear nous algoritmes que puguin desenvolupar nous processos.

## 10.1 Treball Futur

Després del resultat obtingut, cal millorar-lo en alguns aspectes, aquests són els següents:

- Caldria afegir el subprocés de falta de cafè, per tal que el robot pugui procedir en cas que falti cafè.
  - Caldria automatitzar les accions efectuades per comanda, utilitzant els senyals lumínics de la mateixa cafetera utilitzant filtres i algoritmes de la llibreria *OpenCV* o bé, actualitzant els arbres de forma *online*.
  - Caldria afegir un aprenentatge autònom dels arbres de comportament.
  - Caldria instal·lar una GPU per tal que els càculs de detecció d'objectes i persones siguin més ràpids.
  - Caldria entrenar la xarxa neuronal amb més imatges per tal de fer el model més afí a canvis de llum.
- .

## Capítol 11

# AGRAÏMENTS

Primerament, vull agrair la col·laboració de totes les persones que m'han acompanyat al llarg de tot el projecte, sobretot al tutor del present projecte *Néstor Garcia Hidalgo*, sense vosaltres, aquest projecte no s'hagués realitzat.

Seguidament, volia agrair al meu estimat Victor, per totes aquelles hores invertides en aquest projecte, per la gran paciència i motivació donada. A la meva família, pel suport i la confiança i, al meu germà, amics i companys, per aquest gran camí compartit amb il·lusió i desil·lusió que hem seguit i que ha arribat a la seva fi.

## Annex

## Annex A

# Missatges ROS

### A.1 darknet\_ros

#### darknet\_ros\_msgs/BoundingBoxes

```
std_msgs/Header header
uint32 seq
time stamp
string frame_id
std_msgs/Header image_header
uint32 seq
time stamp
string frame_id
darknet_ros_msgs/BoundingBox[] bounding_boxes
float64 probability
int64 xmin
int64 ymin
int64 xmax
int64 ymax
int16 id
string Class
```

#### darknet\_ros\_msgs/BoundingBox

```
float64 probability
int64 xmin
int64 ymin
int64 xmax
int64 ymax
```

```

        int16 id
        string Class

darknet_ros_msgs/CheckForObjectsActionGoal

        std_msgs/Header header
            uint32 seq
            time stamp
            string frame_id
        actionlib_msgs/GoalID goal_id
            time stamp
            string id
        darknet_ros_msgs/CheckForObjectsGoal goal
            int16 id
            sensor_msgs/Image image
            std_msgs/Header header
                uint32 seq
                time stamp
                string frame_id
                uint32 height
                uint32 width
                string encoding
                uint8 is_bigendian
                uint32 step
                uint8[] data

actionlib_msgs/GoalID

        time stamp
        string id

darknet_ros_msgs/CheckForObjectsActionFeedback

        std_msgs/Header header
        uint32 seq
        time stamp
        string frame_id
        actionlib_msgs/GoalStatus status
        uint8 PENDING=0
        uint8 ACTIVE=1
        uint8 PREEMPTED=2

```

```

        uint8 SUCCEEDED=3
        uint8 ABORTED=4
        uint8 REJECTED=5
        uint8 PREEMPTING=6
        uint8 RECALLING=7
        uint8 RECALLED=8
        uint8 LOST=9
        actionlib_msgs/GoalID goal_id
        time stamp
        string id
        uint8 status
        string text
        darknet_ros_msgs/CheckForObjectsFeedback feedback

```

**darknet\_ros\_msgs/CheckForObjectsActionResult**

```

        std_msgs/Header header
        uint32 seq
        time stamp
        string frame_id
        actionlib_msgs/GoalStatus status
        uint8 PENDING=0
        uint8 ACTIVE=1
        uint8 PREEMPTED=2
        uint8 SUCCEEDED=3
        uint8 ABORTED=4
        uint8 REJECTED=5
        uint8 PREEMPTING=6
        uint8 RECALLING=7
        uint8 RECALLED=8
        uint8 LOST=9
        actionlib_msgs/GoalID goal_id
        time stamp
        string id
        uint8 status
        string text
        darknet_ros_msgs/CheckForObjectsResult result
        int16 id
        darknet_ros_msgs/BoundingBoxes bounding_boxes

```

```

std_msgs/Header header
uint32 seq
time stamp
string frame_id
std_msgs/Header image_header
uint32 seq
time stamp
string frame_id
darknet_ros_msgs/BoundingBox[] bounding_boxes
float64 probability
int64 xmin
int64 ymin
int64 xmax
int64 ymax
int16 id
string Class

```

**actionlib\_msgs/GoalStatusArray**

```

std_msgs/Header header
uint32 seq
time stamp
string frame_id
actionlib_msgs/GoalStatus[] status_list
uint8 PENDING=0
uint8 ACTIVE=1
uint8 PREEMPTED=2
uint8 SUCCEEDED=3
uint8 ABORTED=4
uint8 REJECTED=5
uint8 PREEMPTING=6
uint8 RECALLING=7
uint8 RECALLED=8
uint8 LOST=9
actionlib_msgs/GoalID goal_id
time stamp
string id
uint8 status
string text

```

## A.2 openpose\_ros

**openpose\_ros\_msgs/OpenPoseHumanList:**

```

std_msgs/Header header
uint32 seq
time stamp
string frame_id
std_msgs/Header image_header
uint32 seq
time stamp
string frame_id
int32 num_humans
openpose_ros_msgs/OpenPoseHuman[] human_list
int32 num_body_key_points_with_non_zero_prob
int32 num_face_key_points_with_non_zero_prob
int32 num_right_hand_key_points_with_non_zero_prob
int32 num_left_hand_key_points_with_non_zero_prob
openpose_ros_msgs/BoundingBox body_bounding_box
float64 x
float64 y
float64 width
float64 height
openpose_ros_msgs/BoundingBox face_bounding_box
float64 x
float64 y
float64 width
float64 height
openpose_ros_msgs/PointWithProb[25] body_key_points_with_prob
float64 x
float64 y
float64 prob
openpose_ros_msgs/PointWithProb[70] face_key_points_with_prob
float64 x
float64 y
float64 prob
openpose_ros_msgs/PointWithProb[21] right_hand_key_points_with_prob
float64 x

```

```
float64 y
float64 prob
openpose_ros_msgs/PointWithProb[21] left_hand_key_points_with_prob
float64 x
float64 y
float64 prob

openpose_ros_msgs/PointWithProb:
```

```
float64 x
float64 y
float64 prob
```

### A.3 coffee\_machine/State

```
string NodeType
string NodeStatus
```

## Annex B

# Calibració càmera

Per tal de garantir el correcte funcionament de la càmera en un escenari cal calibrar la càmera.

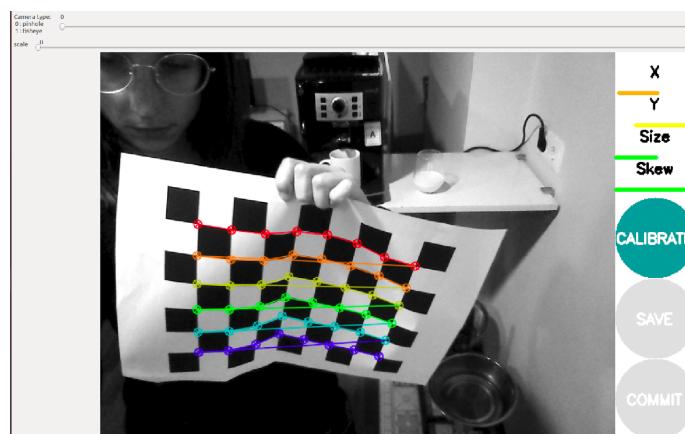
Per fer-ho, cal descarregar-se el paquet *camera\_calibration* inclòs en *ros-perception* [45].

Per calibrar una càmera cal un patró específic, tal com és mostra en la imatge 1.

Utilitzant aquest patró específic, s'executa el paquet *camera\_calibration* assignant-li els següents paràmetres:

- la grandària de la quadrícula : nombre de files per columnes
- quan mesuren els costats del quadrat
- el nom del tòpic de la càmera a calibrar

Una vegada s'executa el node, el patró s'ha de moure per l'espai.



A. 1: Procés de calibració d'una càmera

Una vegada els tres paràmetres X,Y, Size, Skew estiguin en verd, és pot procedir a calibrar la càmera.

Una vegada la càmera estigui calibrada, es guardarà un arxiu a `./ros/camera_info/rgb_Astra_Orbbec.yaml`. Aquest serà el fitxer que utilitzarà la càmera en cada execució.

# Bibliografia

- [1] ABB. Yumi. <https://new.abb.com/products/robotics/industrial-robots/irb-14000-yumi>, Desembre 2020.
- [2] amazon. Delonghi magnifica. [https://www.amazon.es/DeLonghi-Magnifica-ECAM-22-110-B-superautom%C3%A1tica/dp/B004000MU0/ref=sr\\_1\\_1\\_sspa?dchild=1&keywords=cafetera+delonghi+magnifica&qid=1610051609&sr=8-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEzRkxNR0FJMFBGOTdEJmVuY3J5cHR1ZE1kPUEwNDgzOTM3QTZVS0NSVkJMyVlIw](https://www.amazon.es/DeLonghi-Magnifica-ECAM-22-110-B-superautom%C3%A1tica/dp/B004000MU0/ref=sr_1_1_sspa?dchild=1&keywords=cafetera+delonghi+magnifica&qid=1610051609&sr=8-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEzRkxNR0FJMFBGOTdEJmVuY3J5cHR1ZE1kPUEwNDgzOTM3QTZVS0NSVkJMyVlIw), Setembre 2020.
- [3] Varis Autors. C++. <https://es.wikipedia.org/wiki/C%2B%2B>, Octubre 2020.
- [4] Marko Bjelonic. YOLO ROS: Real-time object detection for ROS. [https://github.com/leggedrobotics/darknet\\_ros](https://github.com/leggedrobotics/darknet_ros), Octubre 2016–2018.
- [5] Colab. Subscripció pro a colab. <https://colab.research.google.com/signup>, Setembre 2020.
- [6] The Qt Company. Qt. <https://www.qt.io/>, Setembre 2020.
- [7] Tarun Belagodu Davide Faconti, Esben Damgaard. Groot. <https://github.com/BehaviorTree/Groot>, Setembre 2020.
- [8] Exes. Xml. <https://www.mundolinux.info/que-es-xml.htm>, Setembre 2020.
- [9] Davide Faconti. Behaviortree.cpp. <https://github.com/BehaviorTree/BehaviorTree.CPP>, Agost 2020.
- [10] Tomas Simon Shih-En Wei Hanbyul Joo Gnes Hidalgo, Zhe Cao and Yaser Sheikh. Openpose. <https://github.com/CMU-Perceptual-Computing-Lab/openpose>, Octubre 2020.
- [11] Google. Google drive. [https://www.google.com/intl/es\\_es/drive/](https://www.google.com/intl/es_es/drive/), Setembre 2020.
- [12] google. Google open source. <https://opensource.google/>, Setembre 2020.

- [13] google. Open images dataset. <https://storage.googleapis.com/openimages/web/index.html>, Setembre 2020.
- [14] IRI. Institut de robòtica industrial. <https://www.iri.upc.edu/>, Desembre 2020.
- [15] Software Lab. Què és la gpu o tarjeta gràfica de un ordenador? <https://softwarelab.org/es/que-es-la-gpu-o-tarjeta-grafica-de-un-ordenador/>, Setembre 2020.
- [16] Julia Marsal. Archiu de configuració base del yolo. [https://github.com/easyrobotic/coffee\\_machine/tree/master/cfg](https://github.com/easyrobotic/coffee_machine/tree/master/cfg), Setembre 2020.
- [17] Julia Marsal. Archiu de configuració per entrenar 7 classes. <https://drive.google.com/file/d/1XQX4jaenZkwzNITAeUI1lQXpUNpPH2gk/view?usp=sharing>, Setembre 2020.
- [18] Julia Marsal. Archiu de pesops per detectar 7 classes. [https://drive.google.com/file/d/1-B60hXVvLVSPHw0oF\\_zgo26uZjC1FhQL/view?usp=sharing](https://drive.google.com/file/d/1-B60hXVvLVSPHw0oF_zgo26uZjC1FhQL/view?usp=sharing), Setembre 2020.
- [19] Julia Marsal. Arxius de llançament base del yolo. [https://github.com/easyrobotic/coffee\\_machine/tree/master/launch/darknet\\_ros](https://github.com/easyrobotic/coffee_machine/tree/master/launch/darknet_ros), Setembre 2020.
- [20] Julia Marsal. Darknet folder. [https://drive.google.com/drive/folders/1ZaXWNm6wAw\\_1pyy-cHyETcJy7CSU3IN8?usp=sharing](https://drive.google.com/drive/folders/1ZaXWNm6wAw_1pyy-cHyETcJy7CSU3IN8?usp=sharing), Setembre 2020.
- [21] Julia Marsal. Fitxer behavior\_ree\_node del paquet coffee\_machine. [https://github.com/easyrobotic/coffee\\_machine/blob/master/src/behavior\\_tree\\_node.cpp](https://github.com/easyrobotic/coffee_machine/blob/master/src/behavior_tree_node.cpp), Octubre 2020.
- [22] Julia Marsal. Fitxer xml amb l'arbre de comportament. [https://github.com/easyrobotic/coffee\\_machine/tree/master/xml](https://github.com/easyrobotic/coffee_machine/tree/master/xml), Octubre 2020.
- [23] Julia Marsal. How to train yolov3 using darknet on colab notebook and optimize the vm runtime load times. <https://colab.research.google.com/drive/1GIJYCR1rL5yi5Z8iupo7mVuiuLcPOG6#scrollTo=Pt0Y06QTNyZG>, Setembre 2020.
- [24] Julia Marsal. Node coffe machine. [https://github.com/easyrobotic/coffee\\_machine](https://github.com/easyrobotic/coffee_machine), Octubre 2020.
- [25] Julia Marsal. Resultat dels experiments realitzats. <https://drive.google.com/drive/folders/15Y5ckfj0t-66Ck-eKTsaKYBuWFjE0j34?usp=sharing>, Gener 2021.
- [26] Júlia Marsal. Algorismes d'interacció. <https://drive.google.com/file/d/1b-21FgDTHhXj8HcwzjHzrr-vmJvzYrNV/view>, Desembre 2020.
- [27] Júlia Marsal. yumimotion. [https://github.com/juliamp22/yumi\\_motion](https://github.com/juliamp22/yumi_motion), Desembre 2020.
- [28] Orbbec3d. astra series. <https://orbbec3d.com/product-astra-pro/>, Novembre 2020.

- [29] pccomponentes. Rog strix g. <https://www.pccomponentes.com/asus-rog-strix-g531gu-al018-intel-core-i7-9750h-16gb-256gb-ssd-gtx1660ti-156>, Juliol 2019.
- [30] Josep Redmon. Yolo: Real-time object detection. <https://pjreddie.com/darknet/yolo/>, Agost 2020.
- [31] Joseph Redmon. Darknet: Open source neural networks in c. <https://github.com/pjreddie/darknet>, 2013–2016.
- [32] Joseph Redmon. Classes preentrenades. <https://github.com/pjreddie/darknet/blob/master/data/coco.names>, Setembre 2020.
- [33] Joseph Redmon. Pesos preentrenats. <https://pjreddie.com/media/files/yolov3.weights>, Setembre 2020.
- [34] RO-BOTICA. Bioloid. <https://www.ro-botica.com/Producto/ROBOTIS-PREMIUM-Kit-educativo-Bioloid/>, Desembre 2020.
- [35] ROS. Ros. <https://www.ros.org/>, Setembre 2020.
- [36] ROS.org. Actions. <http://wiki.ros.org/actionlib>, Setembre 2020.
- [37] ROS.org. astra camera. [http://wiki.ros.org/astra\\_camera](http://wiki.ros.org/astra_camera), Novembre 2020.
- [38] ROS.org. Creating a ros package. <http://wiki.ros.org/ROS/Tutorials/CreatingPackage>, Setembre 2020.
- [39] ROS.org. Master. <http://wiki.ros.org/Master>, Setembre 2020.
- [40] ROS.org. Nodes. <http://wiki.ros.org/Nodes>, Setembre 2020.
- [41] ROS.org. Topics. <http://wiki.ros.org/Topics>, Setembre 2020.
- [42] Tzuzalin. labelImg. <https://github.com/tzutalin/labelImg>, Setembre 2020.
- [43] UPC. Grau en tecnologies industrials. <https://www.upc.edu/ca/graus/enginyeria-en-tecnologies-industrials-barcelona-etseib>, Desembre 2020.
- [44] UPC. Ioc. <https://ioc.upc.edu/ca/lioc>, Desembre 2020.
- [45] Joshua Whitley. image pipeline. [https://github.com/ros-perception/image\\_pipeline](https://github.com/ros-perception/image_pipeline), Desembre 2020.
- [46] Wikipedia. C llenguatge de programació. [https://es.wikipedia.org/wiki/C\\_\(lenguaje\\_de\\_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)), Setembre 2020.

- [47] Wikipedia. Cuda. <https://es.wikipedia.org/wiki/CUDA>, Setembre 2020.
- [48] Kevin Zhang. openpose ros. [https://github.com/firephinx/openpose\\_ros](https://github.com/firephinx/openpose_ros), Novembre 2020.