Universitat Politècnica de Catalunya – BarcelonaTech
Facultat d'Informàtica de Barcelona
Enginyeria Informàtica

Millora de la interacció entre humans i robots industrials

Autor: Enric Lamarca Ferrés
Director: Javier Vázquez Salceda
Departament del director: Ciències de la Computació
Especialitat: Computació
Data d'entrega: 23/06/2021
Data defensa: 30/06/2021

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Agraïments

Vull agrair a totes les persones que m'han donat suport durant la realització d'aquest treball de fi de grau, família i amics.

Agrair al director del projecte, Javier Vázquez Salceda, per guiar-me durant tot el procés que comporta un treball de fi de grau.

Agrair a les persones que han col·laborat al projecte, Cecilio Angulo Bahón, Pere Ponsa Asensio i Luis Alejandro Chacón Encalada per les seves aportacions.

Finalment, una menció especial a la meva àvia que en pau descansi per la seva il·lusió i suport.

Abstract

Aquest treball de fi de grau es basa en millorar la interacció entre humans i robots industrials, concretament un braç robòtic de Universal Robots, el UR3 CB-series. La millora consisteix en el disseny i construcció d'una interfície gràfica que permet a l'operador realitzar un conjunt d'operacions i gestionar una sèrie de macro-moviments. A més a més, s'ha investigat la manera de donar semàntica a aquest conjunt d'operacions i macro-moviments, mitjançant una ontologia, amb la idea de que en futurs treballs de fi de grau o treballs finals de màster es pugui afegir interacció amb ordres vocals. Per tals objectius, el projecte s'ha dividit en tres parts.

La primera consisteix en construir una eina visual que permeti programar, parametritzar i simular el *workflow* del procés industrial d'una manera intuïtiva i senzilla per l'usuari. Per tal propòsit s'ha programat una URCap, concretament, un node de programació.

La segona consisteix en construir una eina visual que permeti realitzar un seguiment i control del procés industrial en execució. Per aquesta part s'ha desenvolupat una aplicació Android.

La tercera consisteix en definir una ontologia que doni semàntica al total d'operacions que es poden realitzar amb les eines anteriors. D'aquesta manera es prepara una base semàntica perquè en futurs projectes es pugui afegir control per ordres vocals. Per aquest últim punt s'ha utilitzat l'editor d'ontologies Protégé.

Com a programa base d'on s'han extret els macro-moviments del robot s'ha utilitzat el programa que va programar en Luis Alejandro Chacón Encalada mitjançant Polyscope durant el seu treball de doctorat.

Este trabajo de fin de grado se basa en mejorar la interacción entre humanos y robots industriales, concretamente un brazo robótico de Universal Robots, el UR3 CB-series. La mejora consiste en el diseño y construcción de una interfaz gráfica que permite al operador realizar un conjunto de operaciones y gestionar una serie de macro-movimientos. Además, se ha investigado la manera de dar semántica a este conjunto de operaciones y macro-movimientos, mediante una ontología, con la idea de que en futuros trabajos de fin de grado o trabajos finales de máster se pueda añadir interacción con órdenes vocales. Para tales objetivos, el proyecto se ha dividido en tres partes.

La primera consiste en construir una herramienta visual que permita programar, parametrizar y simular el *workflow* del proceso industrial de una forma sencilla y fácil para el usuario. Para tal propósito se ha programado una URCap, concretamente, un nodo de programación.

La segunda consiste en construir una herramienta visual que permita realizar un seguimiento y control del proceso industrial en ejecución. Para esta parte se ha desarrollado una aplicación Android.

La tercera consiste en definir una ontología que dé semántica al total de operaciones que se pueden realizar con las herramientas anteriores. De este modo se prepara una base para que en futuros proyectos se pueda añadir control por órdenes vocales. Para este último punto se ha utilizado el editor de ontologías Protégé.

Como programa base de donde se han extraído los macro-movimientos del robot se ha utilizado el programa que programó Luis Alejandro Chacón Encalada mediante Polyscope durante su trabajo de doctorado.

This final degree project is based on improving the interaction between humans and industrial robots, specifically a robotic arm of Universal Robots, the UR3 CB-series. The improvement consists in the design and construction of a graphical interface that allows the operator to perform a set of operations and manage some macro-movements. In addition, the way to give semantics to this set of operations and macro-movements has been investigated, by defining an ontology, with the idea that future final degree projects or final master projects can add interaction with vocal commands. For these purposes, the project has been divided into three parts.

The first one is to build a visual tool that allows programming, parameterizing and simulating the industrial process workflow in a simple and user-friendly way. For this purpose, a URCap has been programmed, specifically, a programming node.

The second one is to build a second visual tool that allows monitoring and control of the industrial process in progress. An Android application has been developed for this part.

The third one is to define an ontology that gives semantics to the total number of operations that can be done with the previous tools. In this way, a basis is prepared so that in future projects, control by voice commands can be added. For this last point, Protégé ontology editor has been used.

The program programmed by Luis Alejandro Chacón Encalada using Polyscope during his doctoral work has been used as the base program from which the macro-movements of the robot have been extracted.

Índex

Taula de Figures

1. Introducció

La Revolució Industrial va suposar un gran canvi en la societat i en l'economia del moment. Des de llavors, les indústries han intentat automatitzar al màxim els processos industrials per assolir una producció més ràpida i eficient. D'aquesta manera s'ha pogut satisfer l'exigent i creixent demanda de productes, aliments, vehicles, roba, etc.

Introduir automatització als processos de producció industrial implica introduir màquines, eines, robots que realitzin tasques específiques dins de la cadena de producció. La introducció d'aquests components ha suposat que la interacció entre humans (operadors) i robots industrials sigui d'important rellevància per garantir el correcte funcionament del procés industrial.

Aquest treball de fi de grau es basa en millorar la interacció entre humans i robots industrials, concretament un braç robòtic de Universal Robots [1], el UR3 CB-series. La millora consisteix en el disseny i construcció d'una interfície gràfica que permet a l'operador realitzar un conjunt d'operacions i gestionar una sèrie de macro-moviments. A més a més, s'ha investigat la manera de donar semàntica a aquest conjunt d'operacions i macro-moviments, mitjançant una ontologia, amb la idea de que en futurs treballs de fi de grau o treballs finals de màster es pugui afegir interacció amb ordres vocals.

1.1. Contextualització

El projecte és de modalitat A, és a dir, s'ha realitzat dins del marc de la FIB. És una col·laboració al centre IDEAI-UPC [2] (de l'anglès *Intelligent Data Science and Artificial Intelligence Research Center*) entre varis grups d'investigació. Concretament, és una exploració lateral al projecte de Looming Factory.

Looming Factory [3] uneix diferents grups de recerca i empreses amb l'objectiu d'accelerar la transició cap a la Indústria 4.0. Una nova etapa que es caracteritza per la interconnexió entre màquines i la integració operacional dels treballadors amb l'entorn productiu.

També està relacionat amb altres projectes d'investigació que han estat el punt de partida del propi treball de fi de grau, s'esmenten a continuació:

- **Treball final de màster de Roy Ove Eriksen**: "*Implementation and evaluation of movement primitives and a graphical user interface for a collaborative robot*" [4].
- **Treball de doctorat de Luis Alejandro Chacón Encalada**: "*On Cognitive Assistant Robots for ReducingVariability in Industrial Human-Robot Activities*" (últim article) [5].
- **Treball final de màster de Sergi Ponsà Cobas**: "*Strategies for remote control and teleoperation of a UR robot*" [6].

El braç robòtic de Universal Robots que s'ha utilitzat és un dels que disposa la FIB, situat en un laboratori d'ESAII [7] (Enginyeria de Sistemes, Automàtica i Informàtica Industrial) a l'edifici C5 del Campus Nord.

1.1.1.Definició de conceptes

A continuació es defineixen una sèrie de conceptes que són d'important rellevància i necessaris per la completa comprensió del projecte.

1.1.1.1. Universal Robots

Universal Robots [1] és una empresa dedicada al desenvolupament de braços robòtics, concretament *cobots*. Els anomenats *cobots* són robots col·laboratius preparats i dissenyats per treballar juntament amb humans (operadors). La finalitat principal d'un *cobot* és assistir a operadors humans en espais de treball compartit. D'aquí la importància de tenir un bon canal d'interacció entre humà i robot.

Figura 1: Logotip Universal Robots i UR3 CB-series (Font: [1])

Per aquest projecte s'ha utilitzat un braç robòtic de Universal Robots, concretament el UR3 CB-series que es mostra a la *Figura 1*.

1.1.1.2. GUI (Graphical User Interface)

Una interfície gràfica d'usuari o GUI (de l'anglès *Graphical User Interface*) utilitza elements gràfics, sonors i de control per tal de facilitar a l'usuari la interacció amb un sistema informàtic. L'objectiu principal és aconseguir que l'usuari pugui desenvolupar una sèrie d'accions d'una manera més intuïtiva.

Avui en dia la gran majoria d'aplicacions compten amb una GUI, hi ha molts exemples possibles, entre tots ells, un de molt proper al projecte és Polyscope.

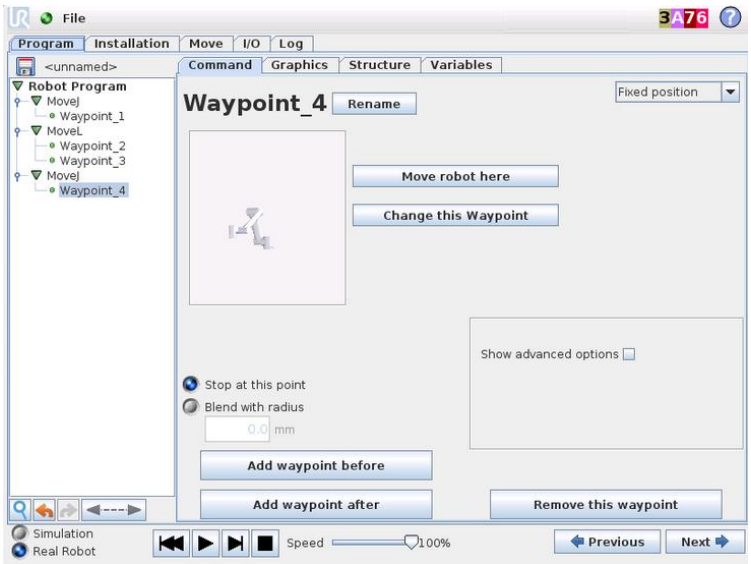

Figura 2: Polyscope CB3 (Font: [8])

Polyscope és la interfície gràfica que ofereix Universal Robots, a la *Figura 2* es pot observar una de les diferents vistes que posseeix. Està carregada a la consola de programació del robot i permet a un usuari amb certa experiència configurar el robot, gestionar entrades i sortides, programar moviments mitjançant nodes de programació que admeten paràmetres, poder visualitzar la seqüencia d'operacions programades en un arbre de programació, poder simular l'execució d'un programa, etc. [8].

En cas de no disposar del robot físic, Universal Robots ofereix un simulador de Polyscope que es pot instal·lar al propi ordinador i tenir disponible les diferents funcionalitats que posseeix, així com les comandes de moviment.

1.1.1.3. URCaps

Universal Robots ofereix la possibilitat d'afegir nous serveis (contribucions) a Polyscope, d'aquesta manera cada empresa o usuari pot afegir certes contribucions depenent de les seves necessitats i tenir un ventall de funcionalitats més ampli i personalitzat. Aquestes contribucions s'anomenen URCaps.

Per programar URCaps, Universal Robots té a disposició dels desenvolupadors un SDK (de l'anglès *Software Development Kit*) que facilita la feina d'integrar correctament les noves contribucions a Polyscope.

El SDK conté documentació de tot tipus, exemples de URCaps, la URCap API (de l'anglès *Application Programming Interface*), scripts per generar el codi base dels diferents tipus de contribucions, etc.

Hi ha 4 tipus generals de contribucions:

- **Nodes d'instal·lació**: són extensions del domini d'instal·lació del robot. Proporcionen a l'usuari una interfície gràfica i afegeixen funcionalitats dirigides a la configuració de l'entorn del robot.
- **Nodes de programació**: són extensions del domini de programació del robot. Proporcionen nous conceptes de programació al flux de programa del robot. Un node de programació també es compon d'una interfície gràfica d'usuari i gestiona el codi URScript que té associat.
- ***Toolbar***: són extensions de la interfície d'usuari general del robot, accessibles des de qualsevol vista de Polyscope. Són útils per visualitzar informació d'estat o controlar equips externs independents de l'estat del robot. Només es poden afegir a versions de Polyscope 5.x.
- ***Daemon***: són extensions que permeten executar un servei al sistema operatiu del robot. Per exemple, un servei programat en Python o C++ que facilita la comunicació amb un dispositiu extern en temps real.

Els tres primers tipus de contribucions són extensions de la interfície d'usuari. Aquests tenen accés a diferents funcions de la URCap API. L'API que ofereix funcionalitats a aquests tipus de nodes es coneix com *Domain* API o API de domini, ja que proporciona accés a les dades i lògica del domini del robot.

L'API de domini es divideix en subtipus (depenent de les funcionalitats que ofereixen i per a quin tipus de contribució van dirigides):

- ***Application* API o API d'aplicació**: proporciona accés a la API rellevant per a la funcionalitat del robot o de l'aplicació. Per exemple, les E/S del robot, les variables de programa, els sistemes de coordenades, etc. Es divideix en dos tipus, que tenen la base d'aquesta API (és a dir, ofereixen totes les funcionalitats de la API d'aplicació) però amb funcionalitats afegides depenent del tipus de node que es vol desenvolupar, d'instal·lació o de programació.
    - ***Installation* API o API d'instal·lació**: inclou funcionalitats específiques pels nodes d'instal·lació.
    - ***Program* API o API de programació**: inclou funcionalitats específiques pels nodes de programació.
- ***User Interface* API o API d'interfície d'usuari**: proporciona accés a la API rellevant per la interfície gràfica d'usuari. Per exemple, teclats.
- ***System* API o API de sistema**: proporciona accés a la API que té informació de sistema del robot. Per exemple, el número de sèrie, la versió de programari o l'idioma del sistema.

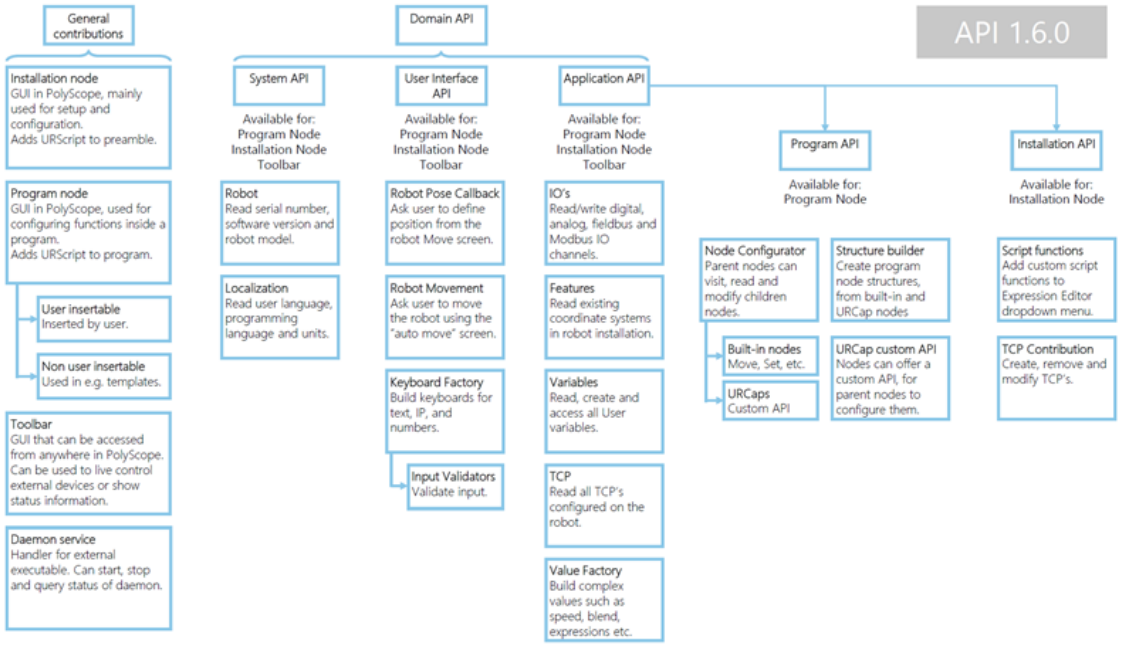

Figura 3: Esquema URCap API (Font: [9])

A la *Figura 3*, es pot observar un esquema que conté els diferents tipus de contribucions i les subdivisions de l'API de domini. Aquets apartat és un resum de [9], que conté informació més detallada.

1.1.1.4. Android

Android és un sistema operatiu basat en una versió modificada del nucli (*kernel*) de Linux i altres programaris lliures de codi obert, dissenyat principalment per a dispositius mòbils. El seu codi font es coneix com AOSP (de l'anglès *Android Open Source Project*) i està sota la llicència de Apache. És un programari lliure de codi obert [10].

Les aplicacions Android són extensions de les funcionalitats dels dispositius. Android ofereix un SDK pel desenvolupament d'aquestes aplicacions. El SDK es compon d'un conjunt d'eines de desenvolupament, entre les quals es poden destacar un *debugger*, llibreries de programari, un emulador de dispositius mòbils basat en QEMU (de l'anglès *Quick Emulator*), documentació, codis exemple, etc. [10].

Android Studio és la IDE (de l'anglès *Integrated Development Environment*) oficial pel desenvolupament d'aplicacions Android i inclou el SDK esmentat; està basada en IntelliJ IDEA [11].

1.1.1.5. Llenguatges implicats

Per una banda, les URCaps es programen en Java. D'altra banda, pel desenvolupament d'una aplicació Android es pot utilitzar Kotlin, Java o C++ [12]. Per obtenir un producte final més homogeni s'ha optat per utilitzar també Java.

Els programes programats mitjançant Polyscope es tradueixen al llenguatge URScript, un llenguatge propi de Universal Robots, el codi generat és el que s'executa al controlador del robot. També hi ha la possibilitat de programar directament en URScript i carregar el programa al controlador del robot [13].

El treball de fi de grau també ha contemplat la possibilitat d'utilitzar Python per a la programació de la GUI o ROS per a la programació del robot. ROS [14] (de l'anglès *Robot Operating System*) és un *framework* per escriure *software* per a robots. És una col·lecció d'eines, llibreries i convencions que tenen com a objectiu simplificar la tasca de crear un comportament complex i robust d'un robot. Els *cobots* de Universal Robots tenen un controlador (*driver*) que possibilita el control d'aquests utilitzant ROS.

1.1.1.6. Ontologia

En l'àrea de la computació, una ontologia es defineix com la representació formal, explícita i general dels conceptes (entitats) d'un domini en concret (en el nostre cas, el domini vindrà determinat pels moviments del robot i les comandes i elements de la interfície gràfica d'usuari per controlar-lo) [15].

Cada un d'aquests conceptes forma una classe dins l'ontologia. Les classes es componen de diferents atributs, característiques i relacions entre altres classes. Aquestes propietats poden tenir restriccions com pot ser el tipus, la cardinalitat, etc. [15].

Els elements d'una ontologia representen predicats, constants, conceptes i relacions que pertanyen a un llenguatge quan s'utilitza per comunicar informació sobre un domini. Per tant, una ontologia es pot considerar un vocabulari, una semàntica pel domini objectiu [15].

Protégé [16] és una eina gratuïta, de codi lliure que permet la construcció de models de domini i aplicacions basades en el coneixement amb ontologies.

1.1.1.7. Norma ISA-101

La Norma ISA-101 defineix una sèrie de terminologies i models recomanats pel desenvolupament d'un HMI (de l'anglès *Human Machine Interfaces*). Es basa en contextualitzar les dades mostrades per millorar el model mental de l'operador en la seva àrea de responsabilitat i en augmentar la percepció de la situació (*Situation Awareness*). Alguns dels beneficis que comporta utilitzar aquest estàndard:

- Augment de la seguretat funcional.
- Millora de la qualitat.
- Una producció major.
- Confiabilitat.

El disseny del HMI ha de suportar les principals tasques de monitorització i control del procés industrial. S'han de considerar els següents principis generals:

- El HMI és una eina efectiva pel control segur i eficient del procés industrial.
- El HMI ajuda en la detecció i diagnosis precoç de situacions anòmales per donar la resposta adequada.
- El HMI s'ha d'estructurar per ajudar a l'operador a prioritzar la resposta davant a múltiples situacions anòmales.
- El HMI ha donar suport a l'operador a l'hora de prendre decisions.
- Percepció de la situació (*Situation Awareness*):
    - Ser conscient del que esta succeint en tot moment.
    - Entendre l'estat del procés industrial.
    - Preveure un estat probable del procés industrial en un futur.
- El HMI ha de donar suport al coneixement de l'operador i a la seva comprensió de l'estat del procés industrial.
    - Quan el procés està funcionant com s'espera, s'ha de mostrar el mínim d'estímuls sensorials.
    - A mesura que el procés es desvia de les expectatives, s'han de proporcionar senyals visuals i/o auditives adequades a la situació.
- Respecte els límits cognitius de l'usuari:
    - Agrupar les dades rellevants.
    - Proporcionar dades rellevants pels rols i responsabilitats de l'usuari.
    - No emplenar vistes amb dades que només es necessiten en certes situacions puntuals.
    - Posar les dades rellevants dins dels objectes, d'aquesta manera l'usuari relaciona la informació més fàcilment.
- Per reduir la càrrega visual de l'usuari es recomana utilitzar colors grisosos i en casos especials (d'emergència) vermells.

La norma ISA-101 és de pagament així que tota la informació s'ha extret d'una presentació subministrada per Pere Ponsa Asensio d'un *webinar* on va assistir.

1.1.2. Problema a resoldre

Com ja s'ha introduït anteriorment, el treball col·laboratiu entre humans i robots industrials està a l'ordre del dia. No tots els operadors tenen els coneixements suficients per programar, configurar o controlar un robot. Doncs, no qualsevol operador està dotat per treballar juntament amb *cobots*. Això és una limitació, ja que obliga als operaris a invertir temps per aprendre els coneixements necessaris per poder realitzar un treball col·laboratiu amb un *cobot*. D'altra banda, les empreses també han d'invertir diners per preparar als seus treballadors.

Però, i si es pogués aconseguir una interfície gràfica intuïtiva i fàcil d'utilitzar? Una interfície que s'adapti al nivell d'experiència de cada operador i només es centri en les operacions que poden ser útils per aquella persona en concret, facilitant-li molt la feina. I si, a part, en un futur es pogués controlar el robot mitjançant ordres vocals, com si d'una persona es tractés?

1.1.3. Actors implicats

Aquest treball de fi de grau és un projecte de recerca per intentar trobar la millor manera de dur a terme una interacció entre humans i robots industrials, doncs, principalment va dirigit al sector de la indústria.

Però cal tenir en compte que els robots cada cop estan més presents al nostre dia a dia en diversos sectors: medicina, investigació, societat, etc. En definitiva, qualsevol sector on hi hagi d'haver un tipus de comunicació entre un humà i un robot sortirà beneficiat si es millora aquest canal de comunicació/interacció, fent-lo més intuïtiu, fluït i adaptat als humans.

Els grups d'investigació implicats també estan interessats en el projecte. Com ja s'ha esmentat anteriorment, aquest treball de fi de grau és una col·laboració entre diversos grups d'investigació del centre IDEAI-UPC [2] (de l'anglès *Intelligent Data Science and Artificial Intelligence Research Center*), concretament una exploració lateral al projecte Looming Factory [3]. A continuació, el conjunt de persones implicades al treball de fi de grau:

- **Javier Vázquez Salceda**: director del treball de fi de grau i membre del grup d'investigació KEMLG [17] (de l'anglès *Knowledge Engineering and Machine Learning Group*) que s'engloba dins de IDEAI-UPC.
- **Cecilio Angulo Bahón**: interessat en el treball de fi de grau i membre del grup d'investigació GREC [18] (Grup de Recerca en Enginyeria del Coneixement) que s'engloba dins de IDEAI-UPC. És el coordinador dels paquets de treball del projecte Looming Factory en els que participa la UPC.
- **Pere Ponsa Asensio**: interessat en el treball de fi de grau i membre del grup d'investigació SIC [19] (Sistemes Intel·ligents de Control) que forma part de CS2AC-UPC [20] (Supervisió, Seguretat i Control Automàtic). Ha aportat la seva expertesa en l'aplicació de robots en entorns industrials.
- **Enric Lamarca Ferrés**: estudiant d'Enginyeria Informàtica, especialitzat en Computació, que ha realitzat el treball de fi de grau.

El centre de recerca IDEAI-UPC engloba diversos grups d'investigació enfocats a la AI (de l'anglès *Artificial Intelligence*) i a la *Intelligent Data Science*, dins de la Universitat Politècnica de Catalunya – Barcelona Tech (UPC) [2]. Entre els diversos grups que engloba, els que estan relacionats amb aquest treball de fi de grau són KEMLG i GREC.

KEMLG té com a principal objectiu l'anàlisi, el disseny, la implementació i l'aplicació de diverses tècniques i metodologies d'Intel·ligència Artificial per donar suport a l'operació i l'anàlisi de comportament de sistemes complexos o dominis del món real [17].

GREC té com a principal objectiu l'estudi i recerca sobre sistemes basats en el coneixement i la seva natura qualitativa, tàcita o intangible. Incorpora àrees d'aplicació de rellevància social que suposen un augment de la qualitat de vida d'éssers, entorns i comunitats [18].

D'altra banda, SIC és un grup dedicat al control automàtic i a la supervisió de sistemes [19]. Forma part de CS2AC-UPC, el Centre Específic de Recerca (CER) en Supervisió, Seguretat i Control Automàtic [20].

Finalment, els autors dels projectes relacionats amb aquest treball de fi de grau, anomenats anteriorment, poden tenir cert interès en els resultats finals del projecte de recerca.

1.2. Justificació

En aquest apartat es valorarà l'estat de l'art actual sobre la temàtica del projecte i es justificarà el seu objectiu principal, comparant-lo amb altres solucions existents.

1.2.1. Estat de l'art

El procés de millorar la interacció entre humans i robots col·laboratius ja fa molt temps que està en marxa, doncs, existeixen diferents alternatives i opcions amb aquesta finalitat.

Com ja s'ha exposat anteriorment, Universal Robots ja ofereix una GUI (Polyscope) on es permet una programació del robot a un nivell més elevat. S'ofereixen diferents nodes de programació on l'usuari pot especificar els paràmetres desitjats d'una manera gràfica i més senzilla que la programació en baix nivell. A part, permet configurar el robot i controlar-lo en temps real, entre d'altres coses.

El treball de doctorat de Luis Alejandro Chacón Encalada, "*On Cognitive Assistant Robots for ReducingVariability in Industrial Human-Robot Activities*" (últim article) [5], ha utilitzat Polyscope per definir una sèrie de macro-moviments (*pick and place*) que podrien ser d'utilitat per aquest treball de fi de grau.

D'altra banda, també tenim el treball final de màster "*Implementation and evaluation of movement primitives and a graphical user interface for a collaborative robot*" [4], amb autor Roy Ove Eriksen, que ha optat per utilitzar ROS per generar trajectòries utilitzant DMPs (de l'anglès *Dynamic Movement Primitives*). Aquest també ha implementat una interfície gràfica que va dirigida cap al principal objectiu d'aquest treball de fi de grau, millorar la interacció entre humans i robots col·laboratius.

Respecte la comunicació entre dispositius externs i el robot, hi ha el treball final de màster "*Strategies for remote control and teleoperation of a UR robot*" [6], amb autor Sergi Ponsà Cobas. Aquest treball consisteix en explorar les diferents opcions que ofereix un robot de Universal Robots per establir una comunicació entre ordinador (o dispositiu extern) i robot i quines són les més adients en cada cas (enviar scripts, controlar el robot en temps real, rebre informació sobre l'estat del robot, etc.). A part, també s'ha programat una interfície gràfica que permet utilitzar alguns dels mètodes d'una manera més intuïtiva per l'usuari.

Tot i que molts d'aquests projectes ja tenen implementada una interfície gràfica d'usuari que en certa manera facilita l'ús del programari que s'ha desenvolupat, les interfícies presentades segueixen molt la línia de les interfícies típiques d'avui en dia, les quals has de tenir certa experiència per utilitzar-les. El treball de fi de grau que es planteja pretén implementar una interfície gràfica molt més intuïtiva i amb una base semàntica, característiques que les interfícies dels projectes esmentats no cobreixen.

Hi ha molts altres projectes enfocats a millorar la interacció entre humans i robots, és una temàtica que s'està explorant exhaustivament, perquè és d'important rellevància en el món on vivim i encara ho serà més en un futur no molt llunyà.

1.2.2.Justificació del projecte

Un cop presentat l'estat de l'art, es pot justificar el projecte. Aquest treball de fi de grau està encarat en donar un pas més, aprofitar el treball realitzat pels diferents projectes que s'han presentat, per oferir una interacció entre robot i humà més intuïtiva, fluïda i amb la possibilitat d'escalar i arribar a una interacció mitjançant ordres vocals, deixant preparada la base semàntica per tal objectiu.

Com a últim punt, remarcar el paper tan important que juga la robòtica col·laborativa avui en dia. Millorar la interacció entre humans i robots implica millorar la producció, l'àmbit de treball, la seguretat, la flexibilitat (qualsevol persona hauria de poder comunicar-se amb robots, encara que no tingui coneixements de programació o robòtica) i molts altres aspectes que justifiquen el fet de necessitar investigar com millorar aquesta interacció.

1.3. Abast

A continuació es definiran els objectius principals del projecte, així com els subobjectius implicats i els requeriments. També es valoraran els obstacles i els riscos que comporta un projecte d'aquestes característiques.

1.3.1. Objectius i subobjectius

El principal objectiu d'aquest treball de fi de grau és investigar quina és la millor manera d'interactuar gràficament amb un robot, cobrint una sèrie d'operacions, ja siguin de programació, control o configuració. D'altra banda, també es vol investigar la manera de donar semàntica a cada una d'aquestes operacions i moviments del robot, definint una ontologia.

A continuació, una llista dels objectius i dels seus subobjectius associats:

1. Implementar una interfície gràfica d'usuari que permeti a un operador realitzar certes operacions, ja siguin de control, configuració o programació del robot de la manera més intuïtiva possible.
    a. Determinar quina de les opcions és més viable i preferible per programar la interfície.
    b. Determinar quin és el conjunt d'operacions que ha de ser capaç de gestionar la interfície, tenint en compte la feina realitzada prèviament pels projectes que ens serviran de base.
    c. Determinar la millor manera de representar la informació, l'estructura del programa del robot i les diferents operacions que permetrà la interfície.
    d. Determinar com adaptar la interfície segons el nivell d'expertesa de l'usuari.
    e. Determinar com reduir la càrrega mental i visual segons la Norma ISA-101 [21].
    f. Determinar la millor manera amb la qual el robot pugui donar *feedback* a l'usuari.
    g. Determinar les KPIs (de l'anglès *Key Performance Indicator*) més rellevants que es podrien mostrar un cop la producció ha finalitzat.
2. Implementar eficientment el conjunt d'operacions que permetrà la interfície.
    a. Relacionar la implementació pròpia amb altres implementacions dels projectes que s'utilitzaran de base.
3. Referent a la programació del robot, determinar des de quin nivell de programació volem partir.
    a. Explorar els avantatges d'utilitzar ROS (programació de baix nivell).
    b. Explorar els avantatges d'utilitzar Polyscope/URScript (programació de nivell mitjà).
    c. Explorar si és convenient utilitzar certs components de les dues opcions anteriors, buscant alguna eina que ens permeti fer *mapping* entre els nodes de programació de Polyscope i el llenguatge ROS.
    d. Aprendre ROS, URScript i a utilitzar la interfície Polyscope, depenent de les opcions escollides prèviament.

4. Poder simular la seqüència de moviments del robot abans de l'execució real. D'aquesta manera l'usuari pot tenir una previsualització del programa.
    a. Depenent de quin mètode utilitzem per a la programació del robot, escollir el simulador més adequat.
5. Determinar la millor manera de realitzar la comunicació entre el robot i l'ordinador (o dispositiu extern) on s'executa el programa.
    a. Estudiar els diferents tipus d'opcions i escollir la més adequada per a cada cas (enviar script al robot, operacions en temps d'execució, des del robot enviar *feedback* a la interfície, etc.).
6. Definir una ontologia que representi el conjunt d'operacions que ofereix la interfície gràfica i el conjunt de moviments possibles del robot.
    a. Buscar ontologies del mateix domini que ens serveixin com a base.
    b. Estendre les ontologies amb un nou mòdul que defineixi els termes específics que no són coberts per les ontologies anteriors.
    c. Modelar les extensions de l'ontologia amb l'editor Protégé [16], que permet editar ontologies i després exportar-les en diferents formats, inclòs el *Ontology Web Language* (OWL).

1.3.2. Requeriments

Altres requeriments que ha de complir el projecte són el següents:

1. El projecte ha d'estar enfocat a poder escalar, afegir millores (com per exemple, el reconeixement d'ordres vocals) i a integrar-se a diferents projectes que els hi podria ser de servei.
2. El codi implementat ha de ser eficient i reutilitzable. L'eficiència és important per aconseguir el nivell de fluïdesa en la interacció que es té com a objectiu.
3. Prioritzar la seguretat del possible usuari del programari. Intentar evitar combinacions de paràmetres que podrien danyar la persona o el propi robot.

1.3.3. Possibles obstacles i riscos

En un inici aquest treball de fi de grau presentava una sèrie de possibles obstacles i riscos. Un dels principals obstacles era posar-se al dia dels diferents projectes que es van desenvolupar i estaven relacionats amb aquest treball de fi de grau. Hi havia el risc de que els elements dels diferents projectes no estiguessin preparats per ser combinats en un mateix sistema, la qual cosa ens obligués a dedicar temps en la creació d'adaptadors o haver de reprogramar alguns components.

Depenent del que es decidís utilitzar per programar el robot (ROS o Polyscope/URScript) es presentaven diferents riscos:

- ROS ofereix programació de baix nivell, el que implica més flexibilitat a l'hora de programar. També, és un dels llenguatges més utilitzats al món de la robòtica (no només de Universal Robots, el que possibilita un àmbit d'aplicació del treball de fi de grau més ample). D'altra banda, ROS és més complicat d'aprendre i d'utilitzar.
- Polyscope/URScript ofereix un nivell de programació més elevat i més senzill d'aprendre. Per contrapartida, tenim el risc de veure'ns massa limitats per les opcions que ofereix Universal Robots (no tindrem la flexibilitat de la programació de baix nivell).

Per aquestes raons es va contemplar la possibilitat d'utilitzar alguna eina que permetés fer *mapping* entre les comandes de moviment de Polyscope i el llenguatge ROS, podent contrarestar els riscos de les dues opcions i aprofitar els avantatges.

A part dels llenguatges de programació del robot, també s'havia d'aprendre a programar interfícies gràfiques en el llenguatge de programació escollit i quina era la manera més apropiada de representar el domini que teníem com a objectiu amb una ontologia utilitzant Protégé [16].

El temps ens podia jugar en contra, però gràcies a les reunions setmanals que es van programar, s'ha intentat complir amb els terminis planejats i aprendre mentre s'avançava el projecte de manera escalonada, centrant els esforços en les temàtiques que eren rellevants per assolir els objectius plantejats.

També, cal destacar la possibilitat de perdre dades o part de la implementació per alguna causa qualsevol, doncs, s'han utilitzat eines per tenir un control de versions, com són GitHub, OneDrive, Google Drive, etc.

Degut a la situació actual de pandèmia, les reunions presencials per realitzar proves del programari amb el robot real es podien haver d'aplaçar. Aquest risc s'ha intentat reduir mitjançant l'ús d'un simulador del robot. De la mateixa manera, haver de treballar a distància podia dificultar l'evolució del projecte.

2. Gestió del projecte

El treball de fi de grau està dividit en tres parts:

a) Construir una eina visual que permeti programar, parametritzar i simular el *workflow* del procés industrial d'una manera intuïtiva i senzilla per l'usuari. Per tal propòsit s'ha programat una URCap, concretament, un node de programació.

b) Construir una segona eina visual que permeti realitzar un seguiment i control del procés industrial en execució. Per aquesta part s'ha desenvolupat una aplicació Android.

c) Definir una ontologia que doni semàntica al total d'operacions que es poden realitzar amb les eines anteriors. D'aquesta manera es prepara una base perquè en futurs projectes es pugui afegir control per ordres vocals. Per aquest últim punt s'ha utilitzat l'editor d'ontologies Protégé.

Aquesta divisió es va definir per tal de garantir l'assoliment de tots els objectius i requeriments que presenta aquest treball de fi de grau.

Com a programa base d'on s'han extret els macro-moviments del robot s'ha utilitzat el programa que va programar en Luis Alejandro Chacón Encalada mitjançant Polyscope durant el seu treball de doctorat.

D'ara en davant, es faran referències a les parts esmentades en aquest apartat com **a)**, **b)**, **c)**.

2.1. Justificació de les tecnologies escollides

En aquest apartat es justifiquen les tecnologies que s'han escollit per a la realització de cada una de les parts d'aquest treball de fi de grau. Cal tenir en compte que les tecnologies s'han escollit en funció del robot amb el que s'ha treballat, el UR3 CB-series de Universal Robots.

Per la part **a)**, s'ha optat per implementar una URCap pels següents motius:

- Per la gran quantitat de material de suport en línia que ofereix Universal Robots sobre la implementació de les URCaps. UR+ és una plataforma que disposa d'un fòrum, documentació, un centre de descàrregues (on es poden descarregar manuals, actualitzacions de software del robot, el SDK pel desenvolupament de URCaps, documentació, etc.), vídeos demostratius, exemples de codi, etc. En definitiva, hi ha una comunitat bastant potent darrere del món de les URCaps.
- Poder instal·lar el *software* a Polyscope (a la pròpia consola de programació del robot), evita tenir la necessitat d'un dispositiu addicional, com seria un ordinador.
- Si l'operador ja té certa experiència utilitzant Polyscope no s'ha d'adaptar en un altre entorn de programació.
- Programar el robot a la seva consola de programació permet realitzar execucions amb el robot real sense la necessitat d'haver de tenir un ordinador a la zona de producció industrial.
- Polyscope ofereix un simulador del robot, doncs, es pot simular el programa abans d'executar-lo al robot real.

- Moltes empreses han desenvolupat URCaps per crear GUIs que facilitin l'ús dels components robòtics que ofereixen (pinces, sensors, etc.). El que indica que és una tecnologia fiable i òptima per l'objectiu que s'ha perseguit des de l'inici del projecte.

Per la part **b)**, s'ha optat per implementar una aplicació Android pels següents motius:

- Un cop programat el *workflow* del programa que s'executarà al robot, no té sentit que l'operador hagi d'estar al costat del robot per poder monitoritzar l'execució del programa. Tampoc té massa sentit que l'operador depengui d'un ordinador, l'operador ha de poder seguir fent altres tasques independents a l'execució del programa, però a la vegada estar informat de l'estat del robot i de l'execució del procés industrial. Doncs, una aplicació per un dispositiu mòbil és la opció més òptima per tal objectiu.
- Permet a l'operador tenir més flexibilitat, mobilitat i no ser tan dependent de l'entorn físic de treball del robot.
- Avui en dia tothom disposa d'una dispositiu mòbil, i la gran majoria tenen Android com a sistema operatiu.

Per la part **c)**, s'ha optat per utilitzar l'editor d'ontologies Protégé pels següents motius:

- És l'eina que es va utilitzar per a la realització d'una pràctica a l'assignatura d'Intel·ligència Artificial, doncs, l'estudiant ja té certa experiència utilitzant-la.
- Té una amplia comunitat d'usuaris i és un dels editors d'ontologies més comuns.

Les tres tecnologies són programaris lliures de codi obert, per tant, no afegeixen llicències ni costos addicionals.

2.2. Anàlisi d'alternatives

La finalitat d'aquest apartat és resumir el procés de selecció de les tecnologies escollides per solucionar el problema objectiu del treball de fi de grau.

Primerament, la idea era implementar una interfície gràfica d'usuari en Python que gestionés tant la programació del *workflow*, com la monitorització del procés en execució. Va ser la primera opció perquè ja s'havien desenvolupat projectes anteriors amb aquest llenguatge de programació.

Per la part de simulació, es va contemplar l'ús de Visual Components, el simulador que ha utilitzat en Luis Alejandro Chacón Encalada al seu treball de doctorat.

Respecte els llenguatges de programació pel robot, es van contemplar Polyscope/URScript i ROS.

Resumint els motius de l'elecció de les tecnologies utilitzades:

- A part del simulador del robot que ja està integrat a Polyscope, els simuladors de tercers com Visual Components, RoboDK, etc. No permeten simular exactament les trajectòries dels diferents tipus de moviment que efectuen els robots de Universal Robots. Doncs, la simulació no seria exacta. Aquest va ser un dels principals motius pels quals es va escollir l'opció d'implementar una URCap (un node de programació que faciliti la programació del *workflow*), instal·lar-la a Polyscope i poder simular el programa amb el propi simulador que ofereix Universal Robots. D'aquesta manera també ens hem estalviat haver de tornar a programar tot el programa en un simulador extern i gestionar el *mapping* entre el codi del simulador extern i el codi que s'acabaria executant al robot real.
- Les URCaps s'implementen en Java, per tant, Python ja va quedar descartat.
- Pel disseny de la interfície gràfica de la URCap, Universal Robots ofereix dues opcions: HTML o Java Swing. Es va optar per Java Swing perquè és molt més flexible i ofereix més possibilitats a l'hora de programar interfícies gràfiques, a banda de que l'estudiant ja tenia certa experiència amb aquesta llibreria de Java.
- Com que treballaríem a nivell de Polyscope amb la URCap, vam decidir que aprofitar el programa que havia definit en Luis Alejandro Chacón Encalada via Polyscope era la millor opció, descartant ROS. Per tant, el codi base d'on s'han extret els macro-moviments que defineixen el *workflow* de la producció industrial és el script generat per Polyscope del programa de Luis Alejandro Chacón Encalada (que està en URScript).
- Per la part de monitorització, com bé s'ha justificat a l'apartat de justificació de tecnologies, vam trobar adient que per tal objectiu, l'operador no hagués d'estar "lligat" al robot, per tant, una aplicació per a dispositius mòbils era la millor opció. Dins dels diferents possibles llenguatges de programació que permet Android pel desenvolupament d'aplicacions, vam optar per Java per poder presentar un producte final més homogeneïtzat.
- Per la definició de la ontologia sempre s'ha tingut com a opció utilitzar Protégé, per tant, no s'han contemplat altres alternatives.

2.3. Metodologia i rigor

En ser un projecte de recerca, la planificació inicial podia estar sotmesa a canvis durant el desenvolupament del projecte. Era necessària una metodologia flexible, que es pogués adaptar a aquests canvis, però que a la mateixa vegada permetés dur un control dels objectius assolits i els que quedaven per complir.

2.3.1. Mètode Àgil

Pels motius plantejats anteriorment, es va decidir que un mètode de treball Àgil [22] era el més indicat. Aquest mètode té en compte els possibles canvis que el projecte pot experimentar durant el desenvolupament per obtenir millors resultats. Es basa en planificar els objectius del projecte en diferents *sprints*, d'aquesta manera es pot fraccionar el projecte en fases i tenir més flexibilitat a l'hora d'abordar canvis en la planificació o les diferents decisions que es van prenent a mesura que s'avança un projecte.

Concretament, s'ha utilitzat Scrum of One que segueix les bases del mètode de treball Àgil, però està pensat per projectes d'una sola persona, la qual ha de prendre la responsabilitat dels diferents rols definits per la metodologia Scrum (normalment, aquests rols estan repartits entre les persones d'un equip de treball: Product Owner, Scrum Master, Development Team) [23].

Juntament amb el director del projecte, es van planificar reunions setmanals (tot i que els *sprints* definits eren de dues setmanes) on es verificava si s'havien complert els objectius especificats a les reunions anteriors, es resumia l'estat del projecte en general, es prioritzava el treball dels propers dies i es replanificava quan era necessari. A part, es podien planificar reunions extres o urgents per certs temes que ho requerissin.

2.3.2. Eines

Les eines utilitzades per dur a terme la metodologia explicada han estat les que s'esmenten a continuació.

Per tal de mantenir un control de versions del *software* que s'ha implementat, s'ha utilitzat GitHub. D'altra banda, per la documentació escrita s'ha utilitzat Google Drive i OneDrive. D'aquesta manera ens hem cobert les espatlles si per qualsevol motiu es perdia alguna part del treball o volíem tornar a una versió anterior.

Per tal de dur un seguiment dels diferents *sprints*, objectius o metes s'ha utilitzat Trello, que s'ajusta molt a la metodologia Àgil. Trello [24] et permet organitzar les tasques en diferents estats (llistes), depenent del punt on es trobin respecte el desenvolupament (pendent, per revisar, tancat, descartat, etc.). També, permet afegir descripcions detallades de cada tasca, així com les subtasques que té associades; les tasques es poden classificar amb etiquetes depenent de la seva tipologia. També, s'ha utilitzat GanttProject [25], un programari lliure que permet crear diagrames de Gantt per així poder tenir un control temporal dels diferents *sprints* i tasques a realitzar.

Per tal de mantenir el contacte amb el director del projecte i amb altres persones d'interès s'ha utilitzat el correu electrònic de Google, Gmail. Per organitzar les reunions setmanals s'ha utilitzat Google Calendar. Per realitzar-les, Google Meet, que permet fer videoconferències amb l'opció de compartir pantalla, xat, etc.

2.4. Planificació temporal inicial

(*Aquest apartat s'explica des del punt de vista que es tenia a l'inici del projecte*)

La durada aproximada del projecte és de 119 dies (4 mesos), compresos entre el 23 de febrer del 2021 i el 21 de juny del 2021. La data de finalització s'ha definit tenint en compte que el període de presentació dels treballs de fi de grau és del 28 de juny del 2021 al 2 de juliol del 2021 segons el calendari acadèmic; la idea és deixar una setmana de marge entre la data final del projecte i les presentacions. D'aquesta manera, es disposarà d'aquesta setmana per afrontar possibles imprevistos, retards o simplement per poder repassar la defensa del treball de fi de grau.

El treball de fi de grau té associats 18 ECTS. La FIB estima la càrrega de treball per part de l'estudiant amb unes 30 hores per crèdit. Això implica 540 hores totals de treball, que dividides entre els 119 dies surten a unes 4,5 - 5 hores de feina diàries (caps de setmana inclosos).

En ser una col·laboració entre diferents grups d'investigació, no hi ha un client en concret que defineixi una data límit específica, per tant, no hi ha restriccions temporals afegides.

2.4.1. Recursos humans i materials

Per desenvolupar aquest treball de fi de grau seran necessaris diferents recursos, tant humans com materials.

- **Recursos Humans**: dins dels diferents grups d'investigació implicats al treball de fi de grau, els principals recursos humans són els que s'esmenten a continuació.
    - Javier Vázquez Salceda **[H1]**: director del treball de fi de grau.
    - Cecilio Angulo Bahón **[H2]**: interessat en el treball de fi de grau i coordinador dels paquets de treball del projecte Looming Factory en els que participa la UPC.
    - Pere Ponsa Asensio **[H3]**: interessat en el treball de fi de grau, aportarà la seva expertesa en l'aplicació de robots en entorns industrials.
    - Enric Lamarca Ferrés **[H4]**: estudiant que realitza el treball de fi de grau.
- **Recursos Materials**:
    - *Hardware* **[M1]**: per a la programació s'utilitzarà un ordinador portàtil amb connexió estable a internet (connexió a través de cable o *Wifi*).
    - Robot **[M2]**: per a la realització de proves reals del programari, s'utilitzarà el robot UR3 CB3-series de Universal Robots. És part del *hardware*, però es considera en un punt a part perquè només s'utilitzarà puntualment.
    - *Software* **[M3]**: només està previst utilitzar programari gratuït: editors de codi com Visual Studio Code o PyCharm; llenguatges de programació com Python, Java, ROS, URScript; el simulador de Polyscope de Universal Robots (disponible a la seva pàgina web); Protégé (disseny de l'ontologia); GitHub (pel control de versions); compte de Google (Google Meet, Google Drive, etc.); Trello (organització i tasques); etc.

L'espai de treball estarà repartit entre el domicili familiar (a Palamós, Girona) i el pis compartit (a Barcelona) de l'estudiant, a banda del laboratori d'ESAII a l'edifici C5 del Campus Nord, on es troba el braç robòtic.

2.4.2.Descripció de les tasques

En aquest apartat es descriuen les diferents tasques que es duran a terme durant el desenvolupament del treball de fi de grau. Cada tasca té associat un identificador, una quantitat d'hores estimada i la descripció que justifica l'objectiu de la tasca i les hores que en principi es planegen invertir.

2.4.2.1. Gestió del projecte **[GP]**

Una bona gestió és essencial per l'èxit d'un treball de fi de grau on el temps és limitat. La gestió del projecte agrupa un conjunt de tasques que, majoritàriament, es duen a terme durant el primer període del desenvolupament del treball de fi de grau.

- Context i Abast **[GP1]** (10h): Abans de començar qualsevol tipus de projecte, s'ha de tenir molt clar quins són els seus objectius i definir el context i l'abast que tindrà. En aquesta tasca també s'ha de justificar el projecte i comparar-lo amb l'estat de l'art actual. D'altra banda, és important indicar quina metodologia de treball s'usarà. Tenir clar tot això és d'essencial necessitat abans de començar a desenvolupar el projecte, per això és la primera tasca que s'ha de dur a terme.

- Planificació temporal **[GP2]** (8h): Un cop es té clar l'abast i el context del projecte, així com els objectius que es volen complir, és hora de determinar una planificació temporal. En aquesta tasca es divideix el desenvolupament del treball de fi de grau en diferents tasques on se'ls hi especifica una estimació temporal i les seves dependències. També es defineixen plans alternatius per contrarestar els riscos que poden presentar-se.

- Pressupost i sostenibilitat **[GP3]** (8h): En aquesta tasca s'estima el pressupost pel desenvolupament del projecte, també se'n justifica la seva sostenibilitat.

- Memòria **[GP4]** (60h): Redacció de la documentació del treball de fi de grau. A part de la gestió de projecte, també s'ha de documentar tota la part tècnica, les decisions preses, les conclusions, etc. És una part indispensable i de molta importància pel treball de fi de grau, és on es demostrarà i deixarà per escrit tota la feina realitzada, per tant, és convenient invertir-hi bastantes hores de treball. La memòria s'escriurà de manera constant durant el període definit de desenvolupament del treball de fi de grau.

- Reunions **[GP5]** (30h): Aquesta tasca inclou les reunions setmanals programades amb el director (cada dimarts, d'aproximadament 1 hora cada una). També, les reunions extres que siguin necessàries amb els altres actors implicats.

- Preparació defensa **[GP6]** (15h): Preparació de la defensa del projecte i la possible demostració amb el robot real. Preparar una bona defensa és important per demostrar al tribunal la feina realitzada al llarg del treball de fi de grau. Cal tenir en compte les hores que es destinaran a la preparació de la possible demostració amb el robot real.

2.4.2.2. Treball previ **[TP]**

Abans de començar amb la implementació d'un primer prototip, és necessari situar-se en l'entorn del projecte, aprendre, estudiar l'estat de l'art i preparar l'entorn de programació.

- Aprenentatge **[TP1]** (40h): Cal tenir una bona base d'aprenentatge abans de començar amb la implementació. Aquesta tasca inclou aprendre a utilitzar Polyscope, un mínim del llenguatge de programació URScript, un mínim de ROS i com utilitzar un robot de Universal Robots mitjançant ROS (si s'opta per utilitzar ROS), com programar interfícies gràfiques amb les llibreries que ofereix el llenguatge escollit per tal objectiu, refrescar la teoria del disseny d'ontologies vista a l'assignatura *GRAU-IA*, etc.

- Estudiar l'estat de l'art **[TP2]** (20h): Estudiar els projectes desenvolupats anteriorment i que podrien servir de base per aquest treball de fi de grau. Definir quines parts podrien ser útils i viables i quines no.

- Preparació de l'entorn de programació **[TP3]** (4h): Instal·lar tot el *software* necessari per començar a programar.

2.4.2.3. Tests i avaluació de resultats **[TR]**

- Tests **[TR1]** (18h): Programar petits tests per fer-se una idea de les diferents opcions que hi ha per desenvolupar el projecte. Aquests tests permetran gestionar i reduir alguns riscos tecnològics (relacionats amb l'ús d'alguns components o la integració entre ells), prendre les decisions pertinents i definir més l'abast del projecte (què s'utilitzarà per programar el robot?, quin llenguatge s'utilitzarà per programar la interfície gràfica?, què es podrà aprofitar dels projectes que s'han desenvolupat anteriorment?, quin simulador s'utilitzarà?, etc.).

- Avaluació de resultats i decisions **[TR2]** (4h): Després d'obtenir els resultats dels tests, valorar-los i decidir. Aquesta tasca també té en compte les possibles reunions entre els actors implicats.

2.4.2.4. Desenvolupar un producte viable mínim **[MVP]**

En la metodologia Àgil és necessari desenvolupar un MVP (de l'anglès *Minimum Viable Product*), d'aquesta manera s'obté una versió bàsica del producte a desenvolupar abans de que s'acabi el termini del projecte, que més endavant s'anirà completant fins obtenir el producte final.

- Disseny GUI **[MVP1]** (6h): Definir quines operacions i funcionalitats bàsiques ha d'oferir el MVP. També, dissenyar la millor manera de mostrar-les dins la GUI.

- Implementació del disseny de la GUI **[MVP2]** (20h): Implementar el disseny de la GUI mitjançant alguna de les llibreries del llenguatge de programació escollit per implementar interfícies gràfiques d'usuari (*front end*).

- Implementació de les funcionalitats de la GUI **[MVP3]** (45h): Implementar les diferents operacions i funcionalitats que ofereix la GUI (*back end*). Possiblement, integrant part del treball dels projectes desenvolupats anteriorment a la implementació pròpia.

- Implementació dels moviments del robot **[MVP4]** (35h): Implementar els diferents macro-moviments del robot que podrà gestionar la GUI, mitjançant ROS, Polyscope o alguna eina de *mapping* entre els dos. En aquesta tasca també es contempla la possibilitat d'aprofitar part del treball dels projectes desenvolupats anteriorment, integrant-la a la implementació pròpia.

- Connexió i simulació **[MVP5]** (30h): Implementar el codi necessari per establir la connexió entre el programa i el simulador o el programa i el robot real. Tampoc es descarta la possibilitat d'aprofitar alguna de les classes ja implementades per projectes anteriors que faciliten l'establiment d'aquesta connexió.

- Definir l'ontologia **[MVP6]** (20h): Definir l'ontologia que representi els diferents conceptes abstrets de les diferents funcionalitats de la GUI i macro-moviments del robot mitjançant Protégé. Aquesta tasca també té en compte el temps invertit en buscar ontologies que es podrien utilitzar com a base del domini.

- Validació **[MVP7]** (11h): Validar el producte viable mínim a mesura que es vagi implementant, mitjançant tests de validació. Aquests tests serviran per poder validar cada una de les operacions i funcionalitats de la GUI implementades, així com els macro-moviments. També es valorarà quant d'intuïtiva és la GUI i si compleix les expectatives inicials.

2.4.2.5. Desenvolupar un producte final **[PF]**

- Completar disseny GUI **[PF1]** (6h): Definir i dissenyar les operacions i funcionalitats afegides que oferirà la GUI del producte final, partint del MVP.

- Ampliar la implementació del disseny de la GUI **[PF2]** (20h): Implementar el disseny de les noves funcionalitats partint de la base del MVP.

- Implementar les funcionalitats de la GUI afegides **[PF3]** (45h): Implementar les funcionalitats i operacions de la GUI afegides partint de la base del MVP.

- Implementar els moviments del robot afegits **[PF4]** (35h): Implementar els macro-moviments del robot afegits partint de la base del MVP.

- Ampliar connexió i simulació **[PF5]** (10h): Si per alguna de les funcionalitats o operacions afegides és necessari afegir algun altre tipus de comunicació entre ordinador-simulador/ordinador-robot, afegir-lo, partint de la base del MVP.

- Ampliar l'ontologia **[PF6]** (20h): Ampliar l'ontologia degudament, partint de la base del MVP.

- Validació final **[PF7]** (20h): A mesura que es vagi completant la implementació del producte final, fer tests de validació de cada un dels components afegits. Degut a que hi haurà més funcionalitats a validar, ocupa més hores que la validació del MVP.

Cal destacar que el nombre d'hores destinades al MVP i al producte final és similar. Això es deu a que el MVP tot i cobrir menys funcionalitats que el producte final, implica més feina ja que es comença el projecte des de 0. Un cop implementat el MVP, el producte final ja disposa d'una base per on començar a implementar les funcionalitats que no cobreix el MVP i que són necessàries pel producte final.

2.4.3.Resum de les tasques i les seves dependències

Id.	Tasca	Hores	Dependències	Recursos
[GP]	**Gestió del Projecte**	**131h**	-	**[H1][H4][M1][M3]**
[GP1]	Context i Abast	10h	-	-
[GP2]	Planificació temporal	8h	**[GP1]**	-
[GP3]	Pressupost i sostenibilitat	8h	**[GP2]**	-
[GP4]	Memòria	60h	-	-
[GP5]	Reunions	30h	-	**[H2][H3]**
[GP6]	Preparació Defensa	15h	**[GP1]-[GP4][PF]**	**[M2]**
[TP]	**Treball Previ**	**64h**	-	**[H4][M1][M3]**
[TP1]	Aprenentatge	40h	-	-
[TP2]	Estudiar estat de l'art	20h	-	-
[TP3]	Preparar entorn de programació	4h	-	-
[TR]	**Tests i avaluació de Resultats**	**22h**	-	**[H4][M1][M3]**
[TR1]	Tests	18h	**[TP3]**	-
[TR2]	Avaluar resultats tests	4h	**[TR1]**	**[H1][H2][H3]**
[MVP]	**Producte viable mínim**	**167h**	**[TP][TR]**	**[H4][M1][M3]**
[MVP1]	Disseny GUI	6h	-	**[H1][H2][H3]**
[MVP2]	Implementació disseny GUI	20h	**[MVP1]**	-
[MVP3]	Implementació funcionalitats	45h	**[MVP2]**	-
[MVP4]	Implementació moviments	35h	**[MVP1]**	-
[MVP5]	Connexió i simulació	30h	**[MVP2]**	-
[MVP6]	Definir ontologia	20h	**[MVP1]**	-
[MVP7]	Validació	11h	**[MVP1]**	**[H1][M2]**
[PF]	**Producte Final**	**156h**	**[MVP]**	**[H4][M1][M3]**
[PF1]	Completar disseny GUI	6h	-	**[H1][H2][H3]**
[PF2]	Implementació disseny GUI	20h	**[PF1]**	-
[PF3]	Implementació funcionalitats	45h	**[PF2]**	-
[PF4]	Implementació moviments	35h	**[PF1]**	-
[PF5]	Ampliar connexió i simulació	10h	**[PF2]**	-
[PF6]	Ampliar ontologia	20h	**[PF1]**	-
[PF7]	Validació final	20h	**[PF1]**	**[H1][M2]**
HORES TOTALS = 540h				

Figura 4: Taula resum tasques (Font: pròpia)

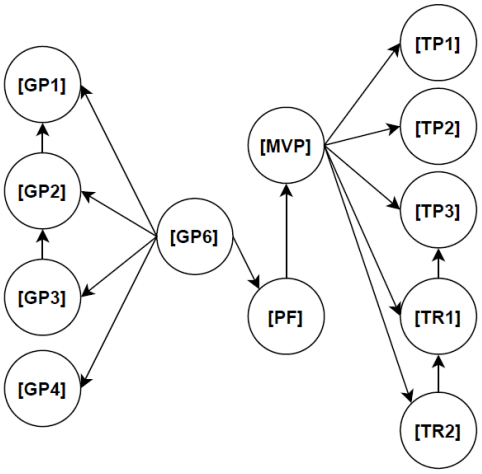

Figura 5: Graf de dependències entre tasques (Font: pròpia)

2.4.4.Diagrama de Gantt

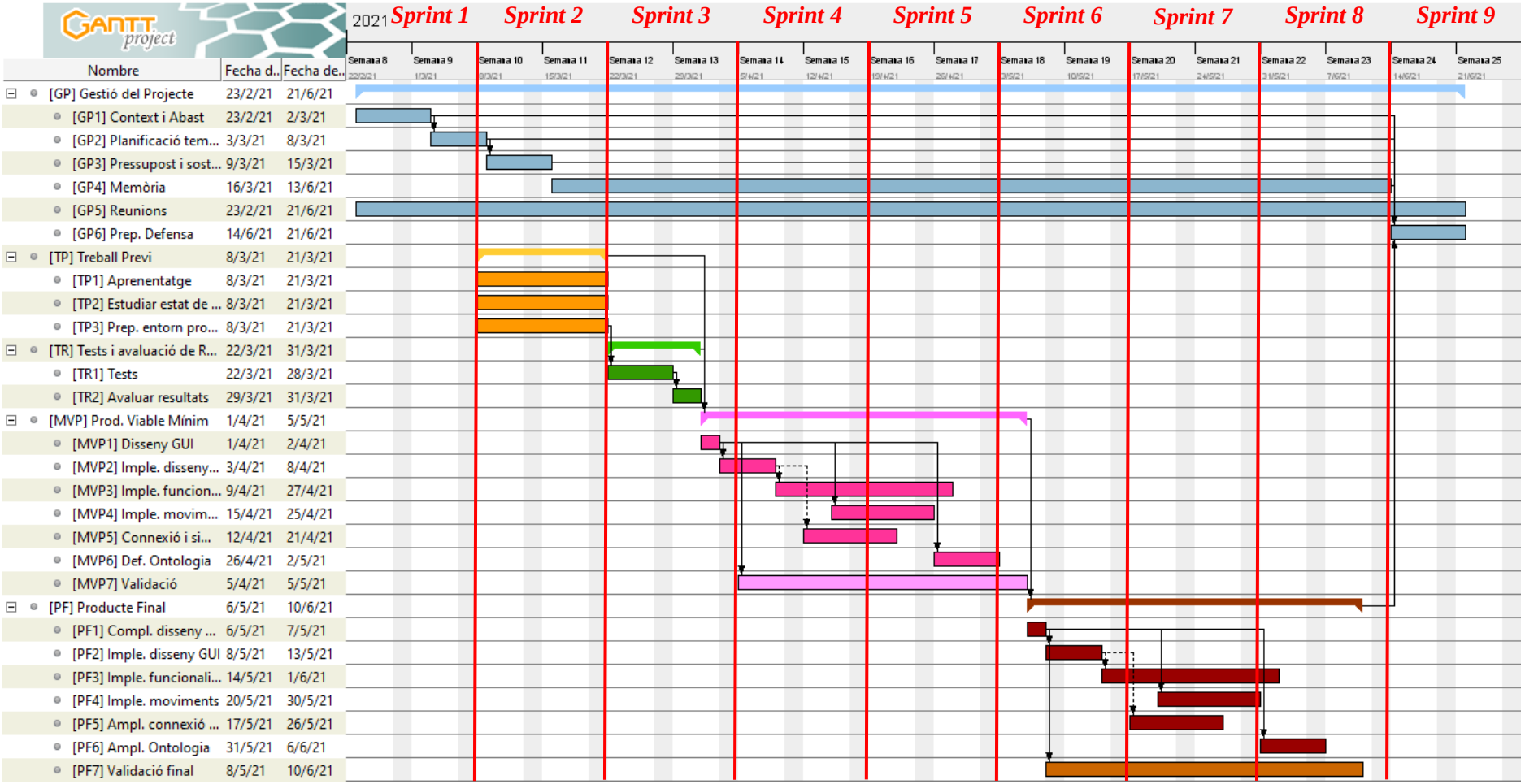

Figura 6: Diagrama de Gantt (Font: pròpia)

2.4.5.Definició dels *Sprints*

Per definir el diagrama de Gantt de la *Figura 6* s'ha utilitzat el programa gratuït *GanttProject* [25]. En aquest apartat es definirà breument l'objectiu principal de cada *Sprint* (*Figura 7*).

Sprint 1	Definir context i abast del projecte, així com la planificació temporal.
Sprint 2	Estimar pressupost, justificar sostenibilitat i completar fase de Treball Previ.
Sprint 3	Realitzar els tests i avaluar els resultats. Plantejar el disseny del MVP.
Sprint 4	Començar implementació del MVP, control intermedi del desenvolupament.
Sprint 5	Finalització del MVP, control final del desenvolupament.
Sprint 6	Plantejar el disseny del producte final i començar a completar el producte.
Sprint 7	Control final de la implementació del producte complet.
Sprint 8	Dedicat a ampliar l'ontologia i millorar-la per deixar-la acabada.
Sprint 9	Preparació de la defensa i la demostració.

Figura 7: Definició dels Sprints (Font: pròpia)

2.4.6.Gestió del risc. Plans alternatius i obstacles

Aquest apartat explica els plans alternatius que s'han plantejat per poder afrontar els diferents obstacles que pot presentar el projecte.

Un dels principals obstacles és la manca de coneixement. Per aquesta raó, s'ha definit el *Sprint 2*, que destina 2 setmanes a l'aprenentatge, abans de començar amb la implementació del producte viable mínim. Aquesta fase d'aprenentatge es basa en cursar cursos online gratuïts sobre els aspectes que no es dominen (ROS, Universal Robots, programar interfícies gràfiques de la millor manera, etc.). A mesura que es vagi completant la fase d'aprenentatge, en paral·lel, també s'estudiarà l'estat de l'art, és a dir, els projectes desenvolupats anteriorment dels quals es podria aprofitar certes parts pel propi treball de fi de grau. Està clar que per aprendre s'ha de practicar, així que en aquesta mateixa fase també es prepararà l'entorn de programació.

Un altre risc és la decisió del programari i eines que s'utilitzaran per desenvolupar el projecte (quins llenguatges de programació, quines eines de simulació, quins programes poden ser útils, etc.). Si es pren una decisió errònia es podria limitar funcionalitats del producte final que en principi eren objectius, o es podrien tenir problemes tecnològics, per exemple de compatibilitat entre programes. Per poder contrarestar aquest altre risc s'ha definit el *Sprint 3*, la realització de petits tests de programari ens estalviarà futurs problemes durant la implementació del producte, a part que facilitaran les decisions de quin programari es preferible i quin no ho és.

El temps del treball de fi de grau és limitat i és molt important poder presentar una versió funcional del programari, encara que no sigui l'esperada perquè el temps no ho ha permès. Per aquesta raó, s'ha escollit un mètode de treball Àgil i el desenvolupament d'un producte viable mínim. D'aquesta manera, si es presenten obstacles o dificultats durant la programació d'una possible versió final, sempre es té la base del MVP (una versió funcional mínima que assegura poder presentar un producte).

En relació al risc presentat al paràgraf anterior, també cal destacar les reunions de control setmanals programades amb el director del projecte. Aquestes permetran detectar els possibles problemes i reaccionar a temps. A més a més, tant el desenvolupament del MVP com el del producte final es divideixen en dos *Sprints*, d'aquesta manera es pot tenir un control més acurat del projecte. Per últim, les hores associades a cada tasca s'han estimat a l'engròs, el qual permet un marge de maniobra més ampli davant dels possibles problemes, obstacles i riscos que es poden presentar.

Actualment vivim una pandèmia a nivell mundial que podria dificultar les validacions i proves del programari implementat que s'hagin de fer de manera presencial a la FIB (on es troba el robot). Per minimitzar aquest risc s'utilitzarà un simulador del robot.

Respecte els recursos, l'ordinador portàtil que s'utilitzarà per a la realització del treball de fi de grau ja té uns 5 anys. Existeix el risc de que pugui fallar en algun moment. Per minimitzar aquest risc es disposa d'un portàtil extra per si fos necessari substituir el principal.

2.5. Planificació temporal final

Aquest apartat explica i justifica els canvis de la planificació temporal inicial i defineix la planificació temporal final.

2.5.1. Justificació dels canvis de la planificació temporal

La planificació inicial es va definir pensant que el producte que es desenvoluparia durant el treball de fi de grau seria una única interfície gràfica programada en Python. Aquesta interfície gràfica s'executaria a l'ordinador i gestionaria tant la programació com la monitorització del procés industrial.

Després de contemplar totes les alternatives existents vam decidir implementar una URCap per a la programació del procés industrial i una aplicació Android per a la monitorització d'aquest mateix, com ja s'ha especificat i justificat en apartats anteriors.

Aquests canvis han afectat directament a la planificació temporal. Doncs, s'han hagut d'invertir més hores en aprenentatge i gestionant diferents problemes tecnològics que han sorgit per la falta d'experiència en els camps de desenvolupament de URCaps i d'aplicacions Android.

Per aquests motius, el desenvolupament del MVP es va allargar un mes més del que s'havia planejat en un inici. Tot i això, el MVP ja presentava un producte bastant avançat:

- Per una banda, la implementació de la URCap estava gairebé completa, quedava pendent afegir algunes funcionalitats més i retocar alguns aspectes d'eficiència, però ja es podien programar certes funcionalitats i gestionar el *workflow* del procés industrial, mitjançant una interfície gràfica d'usuari intuïtiva.
- Per altra banda, l'aplicació Android ja es podia connectar amb el robot via *socket* i rebre paquets d'informació sobre l'estat i configuració del robot.

Pel producte final, i a grans trets, quedava pendent completar la URCap, descodificar la informació que rebia l'aplicació Android i mostrar-la programant una interfície gràfica que seguís la línia i els objectius d'aquest treball de fi de grau, definir l'ontologia del conjunt d'operacions i acabar la memòria.

Està clar, que el temps era just però els avantatges que presentaven les tecnologies escollides valien la pena. S'ha de tenir en compte que el producte final ja comptava amb la base tant de coneixements, com d'implementació adquirida durant el desenvolupament del MVP.

2.5.2.Distribució d'hores per tasca final

A la *Figura 8* es pot observar la taula final de distribució d'hores per tasca.

Id.	Tasca	Hores	Dependències	Recursos
[GP]	**Gestió del Projecte**	**131h**	**-**	**[H1][H4][M1][M3]**
[GP1]	Context i Abast	10h	-	-
[GP2]	Planificació temporal	8h	**[GP1]**	-
[GP3]	Pressupost i sostenibilitat	8h	**[GP2]**	-
[GP4]	Memòria	60h	-	-
[GP5]	Reunions	30h	-	**[H2][H3]**
[GP6]	Preparació Defensa	15h	**[GP1]-[GP4][PF]**	**[M2]**
[TP]	**Treball Previ**	**64h**	**-**	**[H4][M1][M3]**
[TP1]	Aprenentatge	40h	-	-
[TP2]	Estudiar estat de l'art	20h	-	-
[TP3]	Preparar entorn de programació	4h	-	-
[TR]	**Tests i avaluació de Resultats**	**22h**	**-**	**[H4][M1][M3]**
[TR1]	Tests	18h	**[TP3]**	-
[TR2]	Avaluar resultats tests	4h	**[TR1]**	**[H1][H2][H3]**
[MVP]	**Producte viable mínim**	**247h**	**[TP][TR]**	**[H4][M1][M3]**
[MVP1]	Disseny GUI	6h	-	**[H1][H2][H3]**
[MVP2]	Implementació disseny GUI	98h	**[MVP1]**	-
[MVP3]	Implementació funcionalitats	112h	**[MVP2]**	-
[MVP4]	Implementació moviments	5h	**[MVP1]**	-
[MVP5]	Connexió i simulació	15h	**[MVP2]**	-
[MVP6]	Definir ontologia	0h	**[MVP1]**	-
[MVP7]	Validació	11h	**[MVP1]**	**[H1][M2]**
[PF]	**Producte Final**	**76h**	**[MVP]**	**[H4][M1][M3]**
[PF1]	Completar disseny GUI	4h	-	**[H1][H2][H3]**
[PF2]	Implementació disseny GUI	10h	**[PF1]**	-
[PF3]	Implementació funcionalitats	40h	**[PF2]**	-
[PF4]	Implementació moviments	0h	**[PF1]**	-
[PF5]	Ampliar connexió i simulació	7h	**[PF2]**	-
[PF6]	Ampliar ontologia	5h	**[PF1]**	-
[PF7]	Validació final	10h	**[PF1]**	**[H1][M2]**
HORES TOTALS = 540h				

Figura 8: Taula final distribució hores per tasca (Font: pròpia)

2.5.3.Diagrama de Gantt final

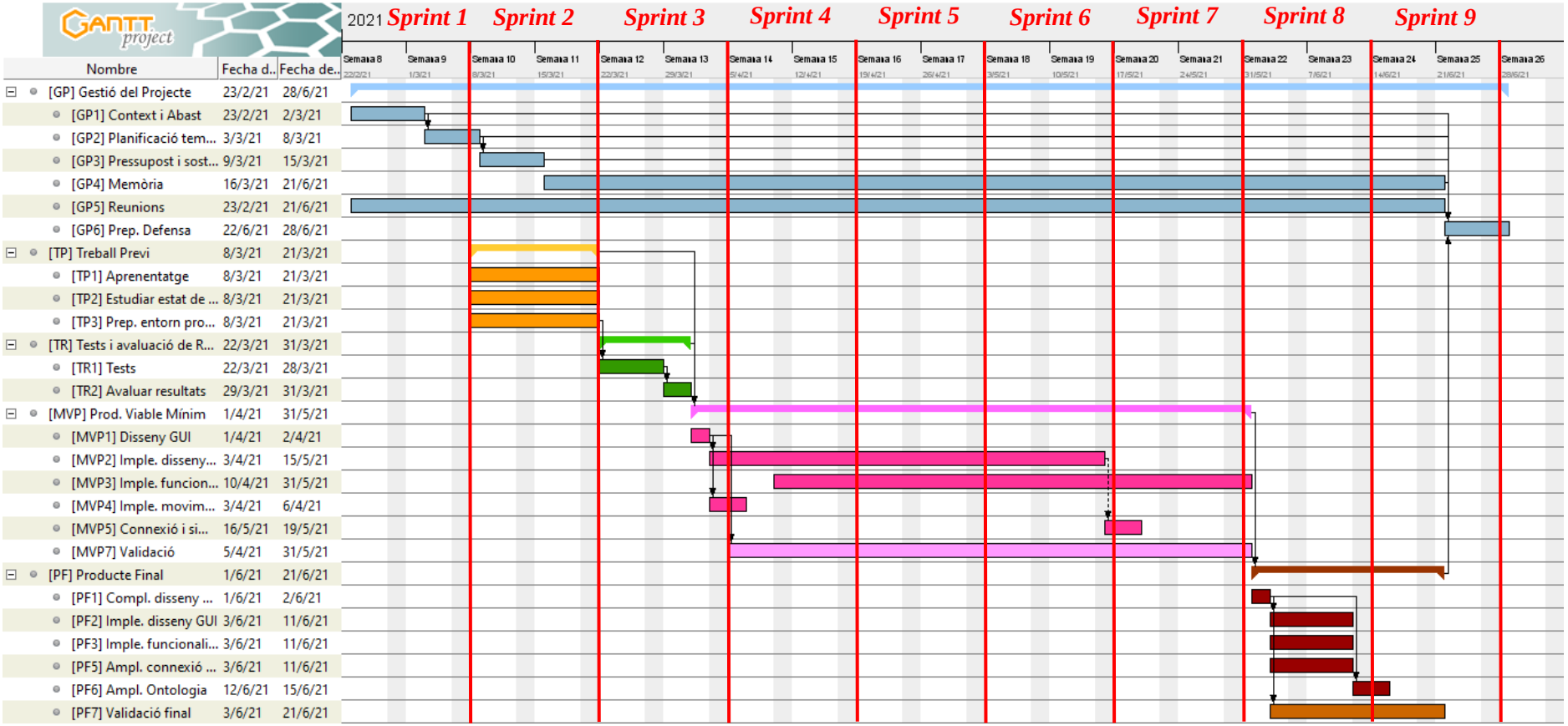

Figura 9: Diagrama de Gantt final (Font: pròpia)

2.6. Gestió econòmica inicial

(*Aquest apartat s'explica des del punt de vista que es tenia a l'inici del projecte*)

En aquest apartat s'identificaran i estimaran els costos que comporta aquest treball de fi de grau, també s'especificarà el control de gestió.

2.6.1. Costos de personal

Per poder estimar els costos de personal, primer, s'ha de determinar el preu per hora de cada un dels rols implicats (cap del projecte, enginyer de desenvolupament de *software* i enginyer de test i validació). Els costos associats a les persones interessades es consideren nuls, ja que només realitzaran reunions puntuals i donaran la seva opinió respecte certs aspectes del projecte. Per estimar el cost per hora de cada un dels rols identificats s'ha utilitzat la guia de mercat laboral 2021 de Hays [26], una empresa de reclutament.

Rol	**Cost per hora (brut)**
Cap del projecte (exercit pel director) **[CPD]**	29,95€/hora
Cap del projecte (exercit per l'estudiant) **[CPE]**	21,35€/hora
Enginyer de desenvolupament de *software* **[EDS]**	18,23€/hora
Enginyer de Test i Validació **[ETV]**	18,75€/hora

Figura 10: Taula cost/hora de personal segons la guia de mercat laboral 2021 de Hays (Font: pròpia)

El sou anual d'un cap de projecte I+D amb més de 10 anys d'experiència és d'uns 57.500€ a Barcelona. Aquest rol serà exercit majoritàriament pel director del treball de fi de grau. Si es suposen 5 dies laborals a la setmana i 8 hores laborals al dia:

57.500€/any * 1 any/12 mesos * 1 mes/4 setmanes * 1 setmana/5 dies lab. * 1 dia lab./8 hores =

29,95€/hora aprox.

El rol de cap de projecte també serà exercit per l'estudiant, ja que aquest és l'encarregat de redactar la memòria (el director s'encarregarà de validar-la). En aquest cas, el sou anual d'un cap de projecte I+D amb 2-5 anys d'experiència és d'uns 41.000€ a Barcelona. Si es suposen 5 dies laborals a la setmana i 8 hores laborals al dia:

41.000€/any * 1 any/12 mesos * 1 mes/4 setmanes * 1 setmana/5 dies lab. * 1 dia lab./8 hores =

21,35€/hora aprox.

D'altra banda, el sou anual d'un enginyer de desenvolupament de *software* I+D amb 2-5 anys d'experiència és d'uns 35.000€ a Barcelona. Aquest rol serà totalment exercit per l'estudiant que realitza el treball de fi de grau. Si es suposen 5 dies laborals a la setmana i 8 hores laborals al dia:

35.000€/any * 1 any/12 mesos * 1 mes/4 setmanes * 1 setmana/5 dies lab. * 1 dia lab./8 hores =

18,23€/hora aprox.

Per últim, el sou anual d'un enginyer de test i validació I+D amb 2-5 anys d'experiència és d'uns 36.000€ a Barcelona. Aquest rol serà exercit majoritàriament per l'estudiant, però també amb la participació del director que validarà les funcionalitats del producte. Si es suposen 5 dies laborals a la setmana i 8 hores laborals al dia:

36.000€/any * 1 any/12 mesos * 1 mes/4 setmanes * 1 setmana/5 dies lab. * 1 dia lab./8 hores =

18,75€/hora aprox.

Cal destacar que el rol de dissenyador (dissenyar el producte *software*) serà exercit pel propi enginyer de desenvolupament de *software* (estudiant que realitza el treball de fi de grau), el cap del projecte (director del treball de fi de grau) i les persones interessades (que com ja s'ha esmentat, no es tindrà en compte el seu sou, ja que participaran de manera molt puntual i interessada).

A continuació, un cop estimat el preu per hora dels diferents rols, s'associa a cada tasca/activitat (identificada a la planificació temporal) un cost, que es calcula segons les hores de treball que implica la tasca i quins rols hi estaran treballant. La taula resultant es pot observar a la *Figura 11*. Cal denotar que les hores que es realitzaran simultàniament entre diferents rols (és a dir, que a la totalitat d'hores de la tasca hauran d'estar presents diferents rols a la vegada) es marquen en cursiva.

Id.	Tasca	Hores	[CPD]	[CPE]	[EDS]	[ETV]	Cost
[GP]	**Gestió del Projecte**	**131h**	**40h**	**91h**	**30h**	**-**	**3.687,75€**
[GP1]	Context i Abast	10h	1h	9h	-	-	222,10€
[GP2]	Planificació temporal	8h	1h	7h	-	-	179,40€
[GP3]	Pressupost i sostenibilitat	8h	1h	7h	-	-	179,40€
[GP4]	Memòria	60h	6h	54h	-	-	1.332,60€
[GP5]	Reunions	30h	*30h*	-	*30h*	-	1.445,40€
[GP6]	Preparació Defensa	15h	1h	14h	-	-	328,85€
[TP]	**Treball Previ**	**64h**	**-**	**-**	**64h**	**-**	**1.166,72€**
[TP1]	Aprenentatge	40h	-	-	40h	-	729,20€
[TP2]	Estudiar estat de l'art	20h	-	-	20h	-	364,60€
[TP3]	Preparar entorn de programació	4h	-	-	4h	-	72,92€
[TR]	**Tests i avaluació de Resultats**	**22h**	**4h**	**-**	**22h**	**-**	**520,86€**
[TR1]	Tests	18h	-	-	18h	-	328,14€
[TR2]	Avaluar resultats tests	4h	*4h*	-	*4h*	-	192,72€
[MVP]	**Producte viable mínim**	**167h**	**7h**	**-**	**156h**	**10h**	**3.241,03€**
[MVP1]	Disseny GUI	6h	*6h*	-	*6h*	-	289,08€
[MVP2]	Implementació disseny GUI	20h	-	-	20h	-	364,60€
[MVP3]	Implementació funcionalitats	45h	-	-	45h	-	820,35€
[MVP4]	Implementació moviments	35h	-	-	35h	-	638,05€
[MVP5]	Connexió i simulació	30h	-	-	30h	-	546,90€
[MVP6]	Definir ontologia	20h	-	-	20h	-	364,60€
[MVP7]	Validació	11h	1h	-	-	10h	217,45€
[PF]	**Producte Final**	**156h**	**8h**	**-**	**136h**	**18h**	**3.056,38€**
[PF1]	Completar disseny GUI	6h	*6h*	-	*6h*	-	289,08€
[PF2]	Implementació disseny GUI	20h	-	-	20h	-	364,60€
[PF3]	Implementació funcionalitats	45h	-	-	45h	-	820,35€
[PF4]	Implementació moviments	35h	-	-	35h	-	638,05€
[PF5]	Ampliar connexió i simulació	10h	-	-	10h	-	182,30€
[PF6]	Ampliar ontologia	20h	-	-	20h	-	364,60€
[PF7]	Validació final	20h	2h	-	-	18h	397,40€
-	**TOTAL CPA[1]**	**540h**	**59h**	**91h**	**408h**	**28h**	**11.672,74€**
-	**TOTAL CPA + SS[2]**	**11.672,74€ * 1,3**					**15.174,56€**

Figura 11: Taula Costos Personal per Activitat (Font: pròpia)

[1] Costos Personal per Activitat.
[2] Seguretat Social, multiplicar per 1,3 el cost total.

2.6.2.Costos generals

En aquest apartat, s'estimaran els costos generals que comporta un projecte d'aquestes característiques.

2.6.2.1. Amortitzacions

- ***Hardware***
    - **Ordinador portàtil**: degut a que només es permet amortitzar el *hardware* en 3-4 anys i el portàtil que s'utilitzarà té més de 5 anys, no es podrà amortitzar (s'hauria de renovar per obsolescència). Però, per allargar la vida útil del portàtil se li ha instal·lat una SSD de 480GB (Sandisk SSD PLUS 480GB [27]). Si es suposa que aquest manteniment allarga la vida útil del portàtil 2 anys, tenint en compte que el projecte estima 540 hores de treball amb l'ordinador i que la SSD val 56,99€: 56,99€/2 anys * 1 any/12 mesos * 1 mes/30 dies * 1 dia/24hores = 0.0033€/hora aproximadament; 540 hores * 0.0033€/hora = **1,80€** amortitzats aproximadament.
    - **Robot UR3 CB3-series**: segons el manual d'usuari [28] que ofereix Universal Robots sobre el robot UR3 CB3-series, la vida útil estimada d'aquest és de 35.000 hores (uns 4 anys). Tenint en compte que el preu és d'uns 23.000,00€ i suposant que s'utilitzarà durant la meitat d'hores destinades a les validacions (unes 15 hores) i durant unes 5 hores per la preparació de la demostració de la defensa del treball de fi de grau: 23.000€/35.000 hores = 0,66€/hora; 20 hores * 0,66€/hora = **13,15€** amortitzats aproximadament.
- ***Software*****:** en principi només s'utilitzarà *software* lliure, per tant, no hi ha costos afegits.

2.6.2.2. Consum elèctric

Actualment, el preu mitjà per dia de la llum està a 0.10342€/kWh [29]. Tenint en compte que tant l'ordinador portàtil com el robot consumiran energia durant el desenvolupament del projecte, aquests costos també s'han de tenir en compte.

- **Ordinador portàtil**: segons les especificacions tècniques del portàtil (HP Notebook - 15-ay050ns) té una potència de 65W. Com s'ha especificat a l'apartat anterior, es suposaran unes 540 hores d'ús de l'ordinador portàtil, això implica un consum de 65W*540h = 35.100Wh = 35,1kWh. Per tant, el preu estimat de consum elèctric és de: 35,1kWh * 0.10342€/kWh = **3,63€**.
- **Robot UR3 CB3-series**: segons el manual d'usuari [28] que ofereix Universal Robots sobre el robot UR3 CB3-series, el consum d'energia del robot executant un programa típic és d'aproximadament 100W. Com s'ha especificat a l'apartat anterior, es suposaran unes 20 hores d'ús del braç robòtic, això implica un consum de 100W*20h = 2000Wh = 2kWh. Per tant, el preu estimat de consum elèctric és de: 2kWh * 0.10342€/kWh = **0,20€**.

2.6.2.3. Espai de treball

L'entorn de treball estarà repartit entre el domicili familiar (a Palamós, Girona) i el pis compartit (a Barcelona) de l'estudiant, a banda del laboratori d'ESAII a l'edifici C5 del Campus Nord, on es troba el braç robòtic.

Tots aquests entorns de treball són compartits, per tant, s'estimarà el cost destinat a l'espai de treball en funció de la tarifa d'un espai de *coworking* a Barcelona. S'ha escollit The Office [30] situat a una zona cèntrica de Barcelona, molt propera a Sants Estació. La tarifa més adequada és la de taula fixa de 150€/mes + IVA i inclou: un estudi fixe, 1Gbps/Ethernet, accés les 24 hores del dia, inclosos caps de setmana, ús il·limitat de les sales de reunions, 2 terrasses.

Per tant, tenint en compte que la durada del projecte és d'uns 4 mesos: 150€/mes + IVA = 181,50€/mes; 181,50€/mes * 4 mesos = **726,00€**.

2.6.2.4. Taula resum costos generals

La *Figura 12* resumeix el total de costos generals presentats.

Partida	**Cost**
Amortitzacions	14,95€
Consum elèctric	3,83€
Espai de treball	726,00€
TOTAL CG[3]	**744,78€**

Figura 12: Taula resum costos generals (Font: pròpia)

2.6.3.Contingències

Com a tot projecte, es poden presentar contratemps o complicacions que encareixin el seu cost final. Per aquest motiu, es presenta una partida de contingències, que pretén donar un marge econòmic al pressupost per poder afrontar situacions d'aquest estil.

Els valors típics de la partida de contingència per projectes de desenvolupament de *software* estan entre el 10% i el 20% del cost final. En aquest cas, s'optarà per un valor intermedi d'un 15% del cost final. Això implica un cost destinat a les contingències de: (CPA + CG) * 15% = (15.174,56€ + 744,78€) * 15% = **2.387,90€**.

2.6.4.Imprevistos

Com s'ha esmentat en la planificació temporal, hi ha riscos i obstacles que poden causar imprevistos al llarg del desenvolupament del projecte.

El principal risc que pot afectar econòmicament al projecte és la possibilitat de que l'ordinador portàtil s'espatlli, degut a que ja té uns 5 anys de funcionament. Aquest imprevist ja s'ha gestionat instal·lant una SSD per allargar la vida útil del portàtil. De primeres, sembla que està donant resultats, però per anar segurs es suposarà la possibilitat d'haver de comprar un ordinador portàtil nou, amb un risc del 30% i una despesa estimada d'uns 600€, el que implica un cost de **180,00€**.

[3] Costos Generals.

D'altra banda, es contempla la possibilitat de que s'espatlli el braç robòtic. Si és el cas, no es reemplaçarà per un de nou, sinó que es repararà (canviant la peça afectada per una de recanvi, reiniciant el *software*, etc.). Suposa un risc del 10% i una possible despesa d'uns 500€ (en el cas de que s'hagi de comprar alguna peça nova o fer venir un tècnic), el que implica un cost de **50,00€**.

La resta de riscos ja es gestionen degudament amb les reunions setmanals, el període d'aprenentatge i tests, l'estimació a l'engròs del nombre d'hores dedicades a la implementació (el que implica una estimació a l'engròs dels costos), l'ús d'un simulador per tal de fer front als riscos presentats per la situació de pandèmia mundial, etc. (com s'ha explicat a la planificació temporal).

Doncs, tenim un total de costos destinats als imprevistos de 180,00€ + 50,00€ = **230,00€**.

2.6.5.Resum pressupost total

A la taula de la *Figura 13* podem observar un resum general dels costos estimats i el seu total.

Partida	**Cost**
CPA	15.174,56€
CG	744,78€
Contingències	2.387,90€
Imprevistos	230,00€
TOTAL	**18.537,24€**

Figura 13: Resum pressupost total (Font: pròpia)

2.6.6.Control de gestió

Per tal de dur a terme un control de gestió s'aprofitaran les reunions setmanals definides. D'aquesta manera, es podrà dur un control constant de possibles desviacions del pressupost i elaborar plans per contrarestar-les i afrontar-les ràpidament.

Per tal de poder calcular i controlar les desviacions s'utilitzaran els següents indicadors numèrics:

Desviació de cost = (CE - CR) * CHR

Desviació de consum = (CHE - CHR) * CE

On:

CE és Cost Estimat, CR és Cost Real, CHR és Consum d'Hores Real i CHE és Consum d'Hores Estimat.

Si les desviacions es deuen als imprevistos que s'han definit anteriorment, s'utilitzarà la partida destinada a aquests mateixos. En cas de que no fos suficient o de que les desviacions provinguessin d'altres imprevistos no contemplats, s'utilitzarà la partida de contingències.

En un projecte de recerca d'aquestes característiques, la planificació inicial es pot veure sotmesa a canvis a mesura que es va desenvolupant el projecte, aquests canvis poden derivar a costos que no es tenien en compte al pressupost inicial. Per aquestes raons, són tan importants les reunions setmanals, el control de gestió que es durà a terme en aquestes mateixes i la partida de contingències (15% de CPA + CG) que permetrà afrontar aquests desviaments.

Si es presentés una desviació exageradament gran, s'hauria de contemplar la possibilitat de redefinir l'abast del projecte perquè s'ajustés al pressupost disponible. Degut a que s'han definit les partides d'imprevistos i contingències, és molt poc probable que succeeixi una desviació suficientment gran com per haver de canviar el rumb del projecte per ajustar-lo al pressupost disponible.

Pel contrari, si les desviacions són a favor, s'utilitzaran les hores extres disponibles per avançar feina. En ser un projecte de recerca, té un abast definit però amb possibilitats d'estendre's, doncs, si es té disponible temps i pressupost que en principi anaven destinats cap a altres tasques que s'han finalitzat gastant menys recursos dels esperats, els recursos sobrants s'invertiran a obtenir un producte final més complet i amb possibles funcionalitats afegides que no es contemplaven en un inici.

Gràcies al robust pressupost i al control de gestió, la basant econòmica del projecte estarà regulada durant tot el període de desenvolupament.

2.7. Gestió econòmica final

Com que els canvis de la planificació temporal no han afectat al total d'hores que s'han invertit per la realització d'aquest projecte, el preu final del pressupost gairebé no ha variat. A la *Figura 14* es mostra la taula final de costos de personal per activitat.

Cal denotar que les hores que s'han realitzat simultàniament entre diferents rols (és a dir, que a la totalitat d'hores de la tasca han estat presents diferents rols a la vegada) es marquen en cursiva.

Id.	Tasca	Hores	[CPD]	[CPE]	[EDS]	[ETV]	Cost
[GP]	**Gestió del Projecte**	**131h**	**40h**	**91h**	**30h**	**-**	**3.687,75€**
[GP1]	Context i Abast	10h	1h	9h	-	-	222,10€
[GP2]	Planificació temporal	8h	1h	7h	-	-	179,40€
[GP3]	Pressupost i sostenibilitat	8h	1h	7h	-	-	179,40€
[GP4]	Memòria	60h	6h	54h	-	-	1.332,60€
[GP5]	Reunions	30h	*30h*	-	*30h*	-	1.445,40€
[GP6]	Preparació Defensa	15h	1h	14h	-	-	328,85€
[TP]	**Treball Previ**	**64h**	**-**	**-**	**64h**	**-**	**1.166,72€**
[TP1]	Aprenentatge	40h	-	-	40h	-	729,20€
[TP2]	Estudiar estat de l'art	20h	-	-	20h	-	364,60€
[TP3]	Preparar entorn de programació	4h	-	-	4h	-	72,92€
[TR]	**Tests i avaluació de Resultats**	**22h**	**4h**	**-**	**22h**	**-**	**520,86€**
[TR1]	Tests	18h	-	-	18h	-	328,14€
[TR2]	Avaluar resultats tests	4h	*4h*	-	*4h*	-	192,72€
[MVP]	**Producte viable mínim**	**247h**	**7h**	**-**	**236h**	**10h**	**4.699,43€**
[MVP1]	Disseny GUI	6h	*6h*	-	*6h*	-	289,08€
[MVP2]	Implementació disseny GUI	98h	-	-	98h	-	1.786,54€
[MVP3]	Implementació funcionalitats	112h	-	-	112h	-	2.041,76€
[MVP4]	Implementació moviments	5h	-	-	5h	-	91,15€
[MVP5]	Connexió i simulació	15h	-	-	15h	-	273,45€
[MVP6]	Definir ontologia	0h	-	-	0h	-	0€
[MVP7]	Validació	11h	1h	-	-	10h	217,45€
[PF]	**Producte Final**	**76h**	**5h**	**-**	**66h**	**9h**	**1.521,68€**
[PF1]	Completar disseny GUI	4h	*4h*	-	*4h*	-	192,72€
[PF2]	Implementació disseny GUI	10h	-	-	10h	-	182,30€
[PF3]	Implementació funcionalitats	40h	-	-	40h	-	729,20€
[PF4]	Implementació moviments	0h	-	-	0h	-	0€
[PF5]	Ampliar connexió i simulació	7h	-	-	7h	-	127,61€
[PF6]	Ampliar ontologia	5h	-	-	5h	-	91,15€
[PF7]	Validació final	10h	1h	-	-	9h	198,7€
-	**TOTAL CPA[4]**	**540h**	**56h**	**91h**	**418h**	**19h**	**11.596,44€**
-	**TOTAL CPA + SS[5]**	**11.596,44€ * 1,3**					**15.075,37€**

Figura 14: Taula final Costos Personal Per Activitat (Font: pròpia)

[4] Costos Personal per Activitat.
[5] Seguretat Social, multiplicar per 1,3 el cost total.

3. Desenvolupament de la URCap

Aquest apartat resumeix els aspectes més rellevants sobre la implementació i el disseny de la URCap que s'ha desenvolupat durant aquest treball de fi de grau.

3.1. Introducció

Com ja s'ha esmentat en apartats anteriors, el principal objectiu de la URCap és oferir una interfície gràfica que faciliti la programació del robot. Per tal objectiu, s'ha implementat un nivell de programació mot alt a partir de blocs parametritzables. Aquests blocs defineixen el *workflow* que representa el flux del programa.

En fer aquest nivell d'abstracció, la programació perd flexibilitat però es limita a les operacions que realment podrien ser d'utilitat per a un operador durant una producció industrial, facilitant-li la feina i reduint el nivell de càrrega visual i cognitiva a l'hora de voler fer un canvi al programa.

Aquest tipus de programació (a nivell de blocs parametritzables) té precedents, un clar exemple és Scratch . Scratch [31] és una eina que permet programar animacions, histories interactives, jocs, etc. A part permet compartir les teves creacions a la comunitat online. Però el que és rellevant per aquest treball de fi de grau és la seva programació, també s'utilitzen blocs parametritzables que constitueixen el *workflow* de l'animació.

La URCAP que s'ha desenvolupat durant aquest treball de fi de grau implementa un **node de programació** que gestiona tot el URScript generat per en Luis Alejandro Chacón Encalada.

Com ja s'ha introduït anteriorment, Universal Robots ofereix un SDK (de l'anglès *Software Development Kit*) per al desenvolupament de URCaps. Pel desenvolupament de la URCap d'aquest treball de fi de grau s'ha utilitzat la versió 1.12 del SDK. El SDK conté un script que et construeix el projecte Maven base per a la URCap. La versió de Java que recomanen utilitzar és la 1.6, doncs, Polyscope 3.x requereix aquesta versió [32]. S'ha utilitzat Eclipse com a IDE (de l'anglès *Integrated Development Environment*), també és la que recomana Universal Robots pel desenvolupament de URCaps. La versió de Polyscope utilitzada per testejar la URCap és la 3.14, tant pel simulador com pel robot real.

La URCap desenvolupada s'anomena EasyProduction, bàsicament perquè la seva finalitat és facilitar un procés de producció industrial.

3.2. Divisió del procés industrial en blocs

Aquest apartat explica com s'ha dividit el programa URScript que representa el procés industrial base en blocs de codi que defineixen el *workflow* del procés. Com ja s'ha mencionat anteriorment, el codi URScript que s'ha utilitzat és el que va generar en Luis Alejandro Chacón Encalada mitjançant Polyscope durant el seu treball de doctorat.

El procés industrial que defineix el programa és un procés d'assemblatge que consta de 3 components que conformen un producte final: la base, el rodament i la tapa. L'estat inicial del procés és la matriu d'assemblatge buida, 4 bases apilades, 4 rodaments apilats i 4 tapes que es troben a disposició de l'operador. A la *Figura 15* es pot veure l'estat inicial del procés:

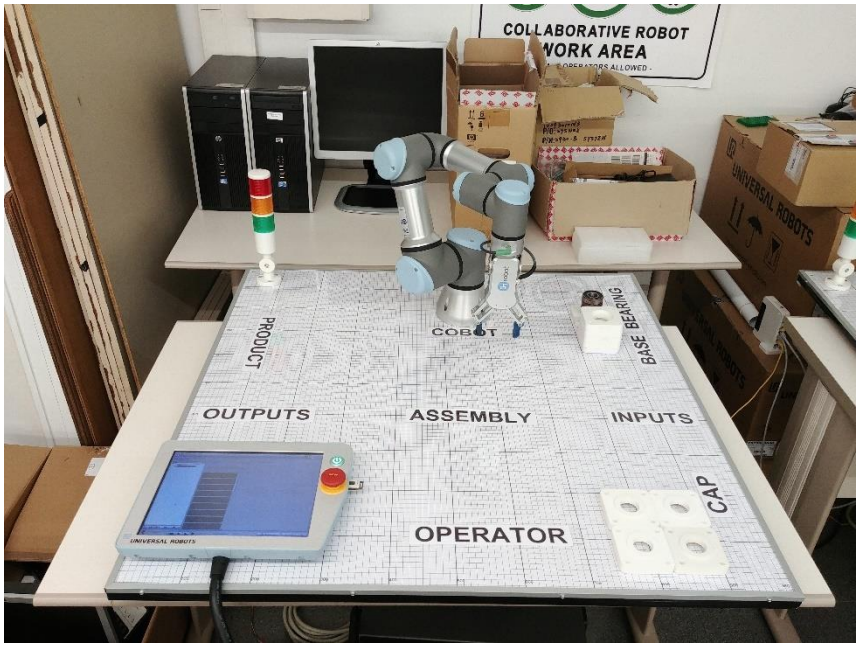

Figura 15: Estat inicial del procés industrial (Font: pròpia)

Durant la primera fase (*Figura 16*) del procés es col·loquen les 4 bases a la matriu 2x2 d'assemblatge. Les bases tenen una cavitat al centre on s'ha d'introduir el rodament, doncs, la segona fase (*Figura 17*) consisteix precisament en introduir els 4 rodaments a les 4 bases que es troben a la matriu d'assemblatge. A la tercera fase (*Figura 18*) entra en joc l'operador (treball col·laboratiu), el robot espera a que l'operador verifiqui que les fases anteriors s'hagin realitzat correctament i col·loqui les tapes sobre les bases, cobrint els rodaments. A la quarta fase (*Figura 19*) els productes ja estan complets a la matriu d'assemblatge, el robot col·loca els productes finals en fila a la zona corresponent als productes.

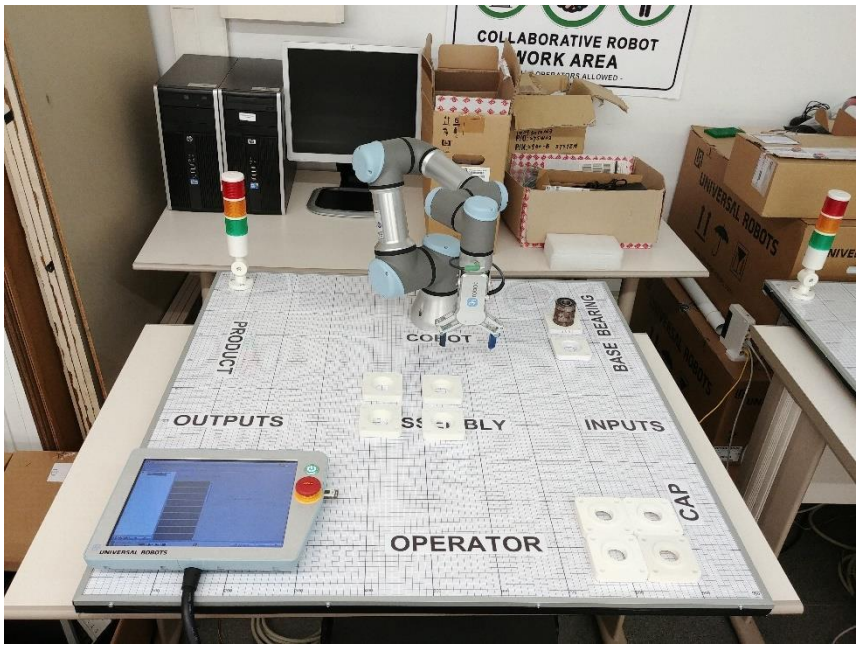

Figura 16: Fase 1 del procés industrial (Font: pròpia)

Figura 17: Fase 2 del procés industrial (Font: pròpia)

Figura 18: Fase 3 del procés industrial (Font: pròpia)

Figura 19: Fase 4 del procés industrial (Font: pròpia)

Un cop explicat el procés industrial es poden definir els blocs que s'han extret del codi font URScript (en aquest ordre):

1. **InitializeVars**: s'inicialitzen les variables globals del programa.
2. **TimerThread**: es defineix i s'inicia el *thread* que gestiona els *timers*.
3. **WriteRegistersThread**: es defineix i s'inicia el *thread* que gestiona les variables que s'escriuen als registres.
4. **ExperimentTimeThread**: es defineix i s'inicia el *thread* que controla el temps de l'experiment que va realitzar en Luis Alejandro.
5. **DefPutBase**: definició de la funció que s'encarrega de col·locar una base un cop desapilada a la matriu d'assemblatge.
6. **DefPutBearing**: definició de la funció que s'encarrega de col·locar un rodament un cop desapilat a la matriu d'assemblatge.
7. **DefPutProduct**: definició de la funció que s'encarrega de col·locar un producte un cop completat a la zona dels productes finals.
8. **While(True)**: bucle principal que s'executa indefinidament fins que l'operador decideix aturar l'execució del programa.
9. **GoToReadyWayPoint**: el robot es mou al punt d'inici on comença el procés d'assemblatge.
10. **If(Bases<4)**: condicional encarregat de controlar que es col·loquen les 4 bases a la matriu d'assemblatge.
11. **DestackBase**: desapila una base.
12. **CallPutBase**: crida a la funció definida prèviament encarregada de col·locar una base a la matriu d'assemblatge.
13. **EndIf**: final del condicional que controla la col·locació de les 4 bases.
14. **If(Bases>=4)**: condicional encarregat de controlar que es col·loquen els 4 rodaments a la matriu d'assemblatge (un cop col·locades les bases).
15. **DestackBearing**: desapila un rodament.
16. **CallPutBearing**: crida a la funció definida prèviament encarregada de col·locar un rodament a la matriu d'assemblatge.
17. **EndIf**: final del condicional que controla la col·locació dels 4 rodaments.
18. **If(Bearings >=4)**: condicional encarregat de controlar que es col·loquen les 4 tapes a la matriu d'assemblatge i llavors es col·loquen els productes finals a la zona corresponent (un cop ja s'han col·locat els rodaments dins les bases).
19. **GetCaps**: notifica a l'operador perquè col·loqui les tapes, bloqueja l'execució del programa.
20. **While(Products<4)**: bucle que s'encarrega de controlar que es col·loquen els 4 productes a la zona de productes finals.
21. **DespalletizeProduct**: agafa un producte final de la matriu d'assemblatge.

22. **CallPutProduct**: crida a la funció definida prèviament encarregada de col·locar un producte a la zona de productes finals.
23. **EndWhile**: final del bucle que controla la col·locació dels 4 productes.
24. **EndIf**: final del condicional que controla la col·locació de les tapes i dels productes.
25. **EndWhile**: final del bucle infinit principal.

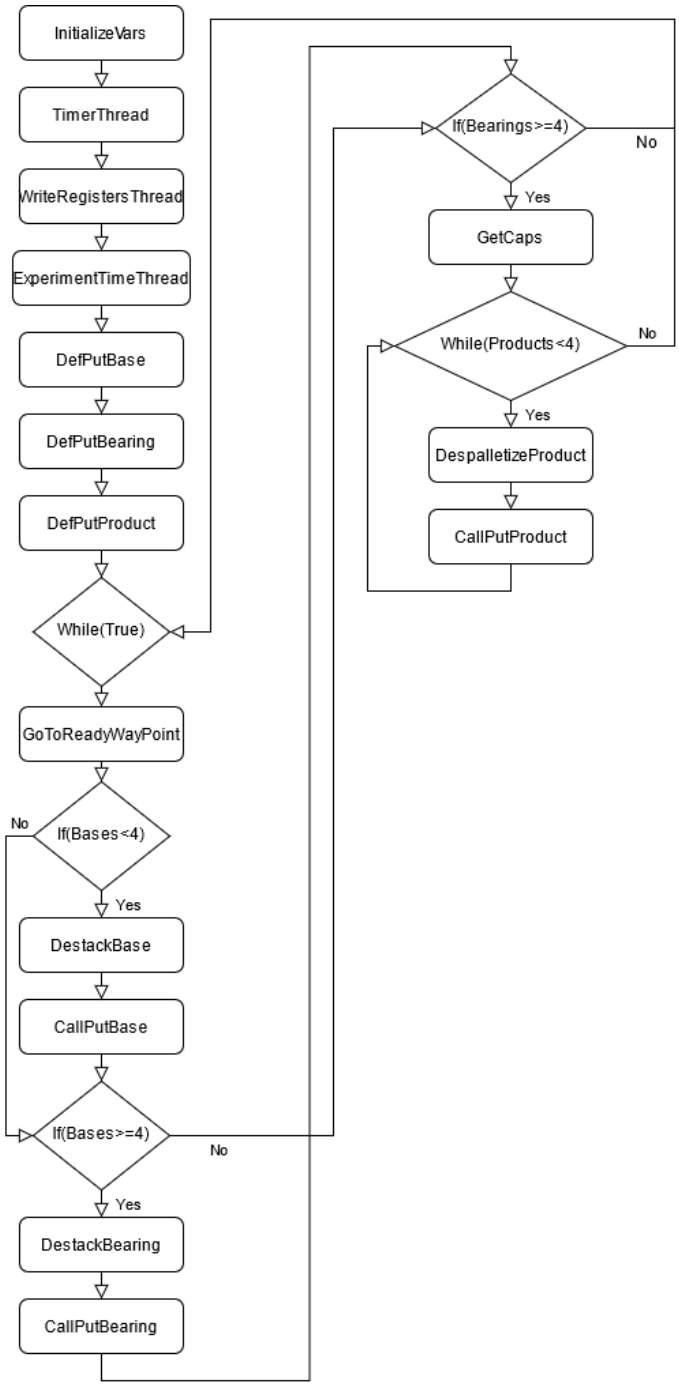

Figura 20: Workflow (Font: pròpia)

A la *Figura 20* podem observar gràficament el *workflow* que defineixen els blocs extrets del procés industrial. Cal destacar que els blocs relacionats amb el flux del procés que indiquen el final d'una condicional o d'un bucle no es mostren a la *Figura 20*, doncs, queden implícits amb les fletxes del diagrama. Tot i això, a l'hora de mostrar el *workflow* a la interfície gràfica sí que s'han afegit aquests blocs per a indicar on acaben els condicionals o bucles definits (no hi ha fletxes que indiquin el flux del programa, per tant s'ha d'indicar explícitament). Cada **End__** indicia el final del condicional o bucle més proper que encara està "obert" (sense tancar).

A banda dels blocs que defineixen el procés industrial, s'han implementat 4 blocs extra de caire demostratiu que són els que l'operador pot afegir o eliminar del *workflow*. La interfície gràfica està preparada per l'addició de més blocs en un futur.

1. **Sleep**: afegeix la comanda URScript sleep() al codi generat. Aquesta comanda para l'execució del procés durant els segons definits per l'usuari.

2. **Popup**: afegeix la comanda URScript popup() al codi generat. Aquesta comanda mostra una finestra emergent amb un missatge. Aquesta finestra pot representar un missatge, un error o un avís (*warning*). També pot bloquejar l'execució del codi fins que no es tanqui.

3. **SetDigitalOutput**: afegeix la comanda URScript set_standard_digital_out() al codi generat. Aquesta comanda defineix el valor d'alguna de les sortides digitals del robot.

4. **SetAnalogOutput**: afegeix la comanda URScript set_standard_analog_out() al codi generat. Aquesta comanda defineix el valor d'alguna de les sortides analògiques del robot.

3.3. Principis d'integració de URCaps a Polyscope

Aquest apartat resumeix els principis bàsics d'integració de URCaps a la interfície gràfica de programació Polyscope [33]. Tota l'explicació està feta des del punt de vista d'una URCap que implementa un nou **node de programació**.

Les URCaps són paquets de programari que s'executen com a un procés fill (secundari) de Polyscope. Polyscope és l'encarregat d'enregistrar la URCap i d'interactuar amb aquesta (durant l'inici de Polyscope o segons les interaccions de l'usuari a la interfície).

Quan Polyscope s'inicia, busca les URCaps que estan instal·lades. Per a cada una d'elles es crida al seu activador (*Activator*). L'activador és l'encarregat d'enregistrar el servei de la URCap. El servei té els mètodes corresponents per crear la interfície gràfica (*User Interface*) i la contribució (*Contribution*) per a cada node, a banda conté propietats generals de la URCap com el nom del node.

La interfície gràfica es pot crear mitjançant HTML o Java Swing i conté tots els elements visuals que ofereix el node. D'altra banda, la contribució és la lògica o el codi de control de cada node. La contribució conté la DataModel, que s'utilitza per guardar la informació de configuració del node corresponent. Quan l'usuari fa una acció a la interfície gràfica, aquesta reporta el canvi a la instància de contribució per actualitzar la DataModel. La contribució també s'encarrega de generar el codi URScript del node de programació quan l'usuari executa el programa.

Quan l'usuari insereix un node de la URCap (en el nostre cas un node de programació a l'arbre de programa de Polyscope), Polyscope crida els mètodes del servei per crear el nou node, aquests mètodes creen la interfície gràfica d'usuari i la contribució del node.

Cal remarcar que cada node de programació té la seva instància de contribució, però tots els nodes del mateix tipus comparteixen la mateixa instància de la interfície gràfica. És a dir, si hi ha diferents nodes del mateix tipus a l'arbre de programa de Polyscope, quan es canvia el node seleccionat, l'única instància de la interfície gràfica ha de carregar la visualització corresponent a les dades de configuració que estan guardades a la DataModel de la instància de contribució del nou node seleccionat.

A la *Figura 21* es mostra un esquema que resumeix el que s'ha descrit en aquest apartat.

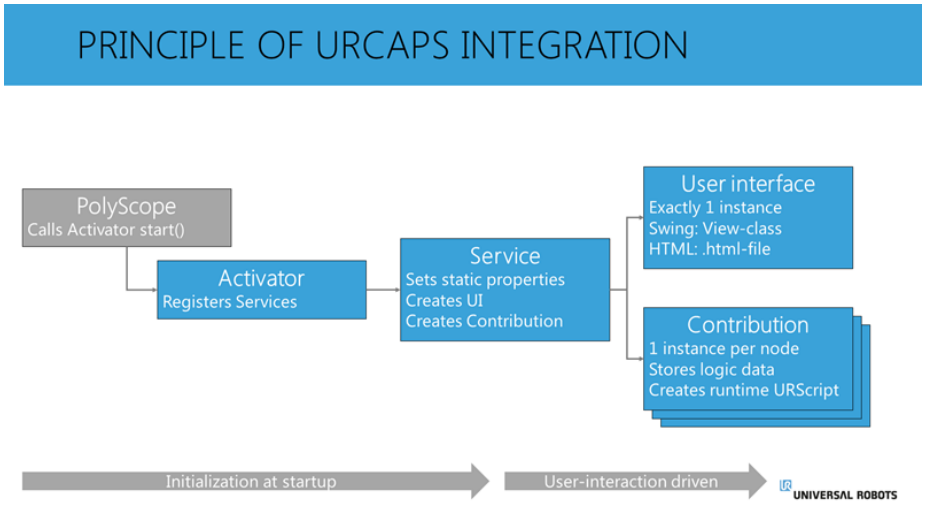

Figura 21: Esquema integració URCaps a Polyscope (Font: [33])

Les classes implementades per a cada un d'aquests elements són:

- Activator, que implementa l'activador del node de programació.
- EasyProductionProgramNodeService, que implementa el servei del node de programació.
- EasyProductionProgramNodeView, que implementa la interfície gràfica del node de programació.
- EasyProductionProgramNodeContribution, que implementa la contribució del node de programació.

3.4. Implementació de la interfície gràfica d'usuari

Aquest apartat explica com s'ha definit la interfície gràfica i com s'han estructurat les classes Java per tal objectiu. Com ja s'ha esmentat en apartats anteriors, després d'analitzar les diferents alternatives hem optat per utilitzar Java Swing per la implementació de la interfície gràfica.

La interfície gràfica es divideix en tres panells principals: el panell **Workflow**, el panell **Opcions** i el panell **Paràmetres**.

La classe Workflow implementa el panell **Workflow** que mostra la seqüència de blocs en què es constitueix el flux del programa. A continuació s'explica com s'han estructurat les classes dels blocs:

- Hi ha una classe general Block que és subclasse de JLabel. Implementa els mètodes que comparteixen tots els blocs, *mouse listeners*, etc. També defineix els atributs que tenen tots els blocs independentment del tipus de bloc.
- Llavors hi ha un conjunt de classes que són subclasses de Block i que agrupen blocs que tenen propietats similars. Aquestes classes contenen atributs i mètodes extra o sobreescrits per a cada conjunt de blocs.
- Finalment, tenim les classes que són subclasses de les classes esmentades al punt anterior. Aquestes classes ja són les que defineixen cada bloc en concret amb els respectius atributs i mètodes.

La classe Workflow és subclasse de JPanel i conté la seqüència de blocs (instàncies de Block, per tant, JLabels) que defineixen el flux del programa.

Llavors tenim la classe OptionsPanel que és subclasse de JPanel i que implementa el panell **Opcions**. Aquest panell conté el conjunt de blocs que es poden afegir al panell **Workflow** fent *drag and drop*. Ull, els blocs que conté aquest panell no són instàncies de Block, són instàncies de la classe Option que és subclasse de JLabel. Les instàncies Option representen el bloc, però la instància Block no es crea fins fer el *drop* del JLabel al panell **Workflow**, afegint un nou bloc al *workflow*. La classe OptionsPanel també conté un JPanel auxiliar on es gestiona l'efecte visual del *drag and drop*.

Finalment, tenim la classe MainPanel que és subclasse de JLayeredPane i que gestiona la col·locació de tots els panells. La classe JLayeredPane és com JPanel però permet afegir components visuals a diferents nivells (*layers*). Doncs, és molt útil per a la gestió del *drag and drop*. La classe MainPanel col·loca la instància de Workflow i el JPanel auxiliar per fer *drag and drop* que té OptionsPanel a la capa DRAG_LAYER, mentre que col·loca la instancia OptionsPanel a la capa DEFAULT_LAYER. D'altra banda, MainPanel té un JPanel que és el

panell **Paràmetres**, també afegit a la DEFAULT_LAYER. Aquest panell s'actualitza depenent del bloc seleccionat al *workflow* amb la interfície gràfica dels paràmetres corresponents a aquest (cada instància Block té el seu panell de paràmetres, un JPanel, que es col·loca al panell **Paràmetres** quan es selecciona).

Les classes Workflow, OptionsPanel i MainPanel segueixen el patró *singleton* (només una instancia de cada classe durant l'execució, seguint la lògica de les URCaps, només una instància de la interfície gràfica).

Doncs, la classe EasyProductionProgramNodeView té un mètode buildUI() que només li cal mostrar la instància MainPanel.

Finalment, remarcar que els 3 panells estan preparats per fer *scroll* i que la interfície gràfica d'usuari està pensada per ser compatible tant amb accions tàctils com utilitzant el ratolí.

3.5. Implementació de la contribució

La contribució està implementada a la classe EasyProductionProgramNodeContribution. Ha de gestionar la generació de codi URScript (a partir del *workflow* definit a la interfície gràfica) i la DataModel on es guarden les dades de configuració del node per quan s'hagi de carregar la seva visualització, sigui corresponent a la seva configuració actual.

Per la generació de codi, cada una de les classes Block, definides a l'apartat anterior, té un mètode que segons els paràmetres definits per l'usuari mitjançant la interfície gràfica, et retorna el codi URScript corresponent. Doncs, la classe EasyProductionProgramNodeContribution només li cal accedir a la instància *singleton* de la classe Workflow i cridar el mètode generateCode() que recórrer tots els blocs del *workflow*, generant el codi de cada un d'ells, donant com a resultat el codi URScript final de tot el programa. Cal destacar que la generació de codi té en compte la correcta indentació i si es genera el codi per simular el programa al simulador del robot de Polyscope o per l'execució real. Aquest últim punt és necessari perquè el simulador del robot no té cap eina, no pot executar comandes de detecció de força, etc. Doncs, totes les comandes que el simulador no pot efectuar s'han de treure o substituir per d'altres que simulin l'execució real del programa (el robot real sí que pot executar totes les comandes, tant les relacionades amb l'eina, com les de força). El codi generat ja inclou el while(True) principal, així que per una correcta indentació es recomana treure el while(True) que genera Polyscope per defecte.

La gestió de la DataModel és més complexa. La DataModel funciona com un diccionari, cada entrada a la DataModel té una clau i un valor. Té un mètode set() per actualitzar el valor corresponent de la clau passada per paràmetre i un altre, get() per obtenir el valor corresponent de la clau passada per paràmetre. D'aquesta manera, es pot guardar la configuració del node a partir de parelles clau-valor i després tornar a carregar la visualització a partir d'aquestes dades.

La classe EasyProductionProgramNodeContribution té dos mètodes molt importants:

- openView(), es crida quan s'obra la interfície gràfica del node i s'ha d'encarregar d'actualitzar-la depenent de la configuració guardada a la DataModel (del node actual).
- onChangeInWF(), es crida quan l'usuari fa una acció a la interfície gràfica que afecta a les dades de la DataModel i s'han d'actualitzar.

Així doncs, pel node de programació que s'ha implementat durant aquest treball de fi de grau es necessita guardar el *workflow* i de cada bloc els seus paràmetres. Quan s'afegeix algun bloc,

s'elimina o es canvia algun dels seus paràmetres s'ha d'actualitzar la DataModel corresponentment.

Universal Robots no permet guardar instàncies de classes pròpies a la DataModel, així doncs, no es poden guardar les instàncies dels blocs en un *array* i actualitzar-lo quan alguna cosa canviï. S'haurien de guardar per cada bloc els paràmetres per separat en diferents entrades a la DataModel, a part de l'ordre dels blocs al *workflow*, una gestió excessivament complicada.

Per tal d'evitar complicar excessivament el codi i dificultar l'addició de blocs en futurs projectes (un dels requeriments és programar el codi d'una manera que sigui fàcil d'escalar) s'ha gestionat diferent.

No es pot guardar un *array* d'instàncies de Block, però sí es pot guardar un *array* de Strings, més concretament, dels Jsons de les instàncies. El problema és que no es poden generar Jsons d'instàncies d'elements visuals o que implementen *mouse listeners*, o almenys quan es va intentar donava errors. Així que s'han hagut d'implementar classes auxiliars que només contenent la informació de cada Block que s'ha de guardar a la DataModel, anomenades BlockData. Cada una de les subclasses de Block té una subclasse de BlockData que la representa.

Doncs, mitjançant la llibreria gson es generen els Jsons de les instàncies BlockData que constitueixen el *workflow* i es guarden en un *array* de Strings a la DataModel. Per tal de poder recuperar les instàncies de BlockData corresponents, es guarda un segon *array* de Strings amb els tipus de cada un dels BlockData. Per recuperar les instàncies Block a partir de les de BlockData, aquestes últimes implementen el mètode getBlockInstance() que té com a objectiu retornar una instància Block amb les dades corresponents a la instància BlockData que la crea.

D'aquesta manera el mètode openView() llegeix el *array* de Jsons dels BlockData de la DataModel, també el *array* dels tipus de BlockData; amb aquesta informació ja pot generar les instàncies BlockData i d'aquestes adquirir les instàncies Block que s'afegeixen al *workflow* per visualitzar-les. Aquesta funció té **cost lineal** respecte el nombre de blocs del *workflow*.

Com adquirir les instàncies BlockData a partir dels Jsons? Hi ha dues opcions: utilitzar la pròpia llibreria gson o *parsejar* "manualment" les Strings Jsons i crear les instàncies de BlockData a partir dels atributs extrets. Després de realitzar una investigació i de trobar alguns problemes d'eficiència que es comenten posteriorment en aquest document, es va optar per la segona opció, doncs, la deserialització de Jsons de la llibreria gson no és gaire eficient. D'aquesta gestió s'encarrega la classe MyStringDeserialization que mitjançant la llibreria regex s'ha definit una expressió regular per extreure els atributs guardats al Json i poder instanciar el BlockData corresponent.

Per actualitzar el *array* de Jsons guardat a la DataModel quan la interfície gràfica rep una acció de l'usuari (afegir bloc, eliminar bloc, seleccionar bloc, canvi del valor d'un paràmetre d'un bloc, *undo, redo*) d'una manera eficient s'ha implementat el mètode onChangeInWF() de la següent manera:

- El mètode rep com a paràmetre una llista de les posicions del *workflow* que han canviat degut a l'acció de l'usuari (principalment és gairebé sempre un bloc només, però en el cas de canvi de selecció de bloc s'han de passar dues posicions, la del nou bloc seleccionat i la del bloc que deixa d'esta seleccionat pel canvi de selecció).

- El mètode registra el canvi en un UndoRedoManager (que s'adquireix a partir de la URCap API) a través de la crida recordChanges(), que se li passa per paràmetre una instància d'una classe que implementa la interfície UndoableChanges, en aquest cas MyUndoableChanges que és una classe interna de

EasyProductionProgramNodeContribution i la seva constructora té com a paràmetre la llista de posicions que han canviat al *workflow* (el paràmetre que se li passa a OnChangeInWF()).

- MyUndoableChanges té la llista de posicions del *workflow* que s'han d'actualitzar. I implementa el mètode de la interfície UndoableChanges executeChanges().
- El mètode executeChanges() comprova quin tipus de canvi s'ha efectuat: canvi en els paràmetres d'un bloc (inclou el canvi de selecció), s'ha afegit un bloc o s'ha eliminat un bloc. Per cada un dels 3 tipus de canvis té **cost lineal** respecte el nombre de blocs del *workflow*, doncs, els *arrays* de Java no es poden redimensionar, així que n'has de crear un de nou i reomplir-lo. En el cas de canvi d'un paràmetre d'algun dels blocs del *workflow* no cal redimensionar l'*array* però se'n crea un de nou amb les mateixes dimensions per evitar problemes de referència (bastant complicat d'explicar, per més detalls mirar documentació del codi).
- Passar per paràmetre la posició dels blocs que s'han d'actualitzar evita haver de generar els Jsons de totes les instàncies de BlockData de nou i només generar-los per les instàncies de BlockData que han canviat.

Gestionar degudament la DataModel és molt important perquè a part de contenir la configuració actual del node, les dades que es guarden quan es guarda el programa són les que conté la DataModel, i quan es carrega el programa guardat s'executa openView() amb aquestes dades perquè la visualització de la interfície gràfica concordi amb la configuració guardada prèviament. D'altra banda, els *undo/redo* també funcionen d'una manera similar, al fer *undo* o *redo* es carreguen les dades de la DataModel anteriors o posteriors, respectivament, i llavors es crida a openView() per actualitzar la visualització amb les dades corresponents. Quan es fa un canvi, només s'actualitza la DataModel, no cal cridar a openView(), perquè el canvi visual ja l'ha gestionat directament les classes implementades que construeixen la interfície gràfica (*mouse listeners*, *action listeners*, *change listeners*, etc.).

3.6. Disseny de vistes

Com ja s'ha especificat en apartats anteriors, la interfície gràfica implementada consta de 3 panells. El panell **Workflow** que conté la seqüència de blocs que defineixen el flux del procés, el panell **Opcions** que conté els blocs que es poden afegir o eliminar al *workflow* i, finalment, el panell **Paràmetres** que conté la interfície gràfica per ajustar els paràmetres del bloc seleccionat (si no hi ha cap bloc seleccionat, el panell està buit).

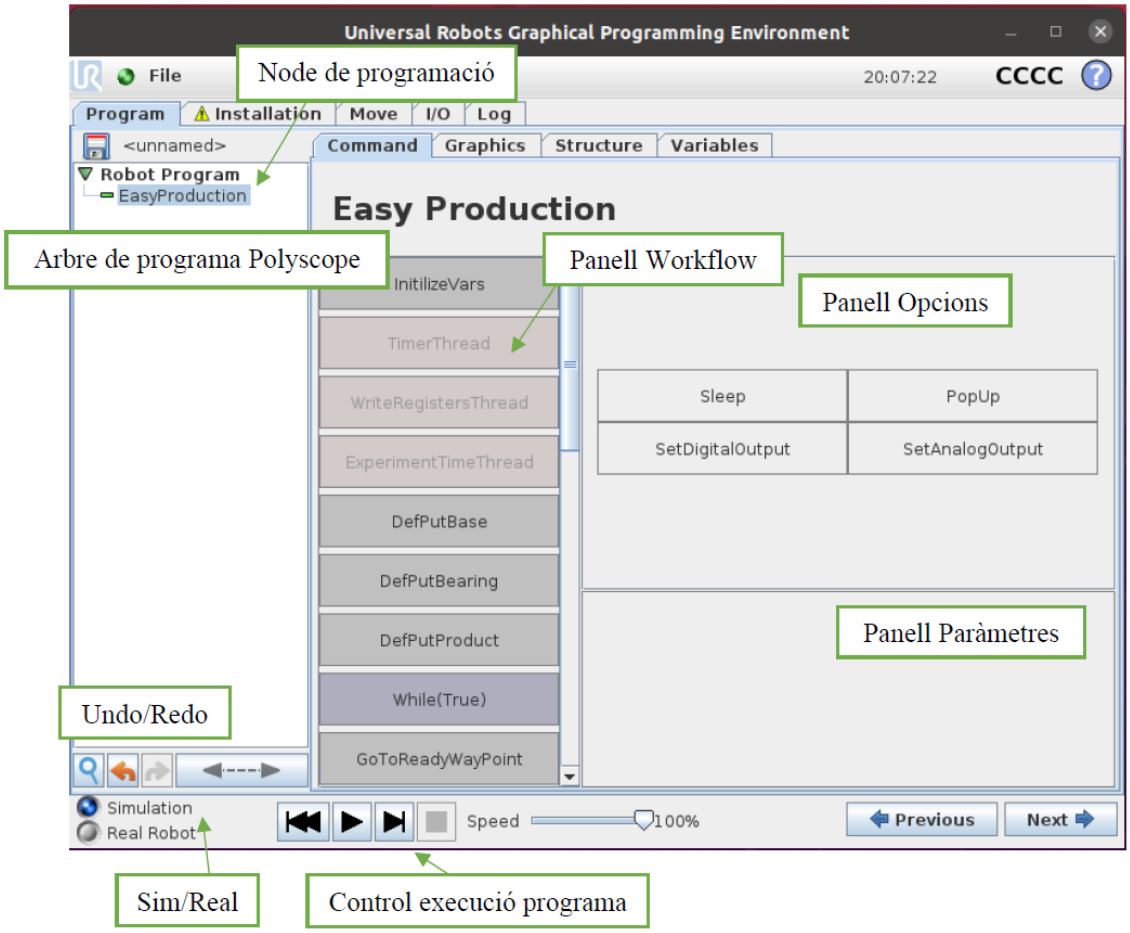

Figura 22: Polyscope mostrant la GUI del node de programació desenvolupat (Font: pròpia)

A la *Figura 22* es pot observar la interfície de programació que ofereix Universal Robots, Polyscope. Com es pot veure a la *Figura 22* a l'arbre de programa de Polyscope només hi ha el node de programació que s'ha implementat, aquest únic node gestiona tot el procés industrial programat per Luis Alejandro Chacón Encalada mitjançant la interfície gràfica Polyscope. Quan es selecciona el node, a la pestanya Command, es troba la interfície gràfica que té associada. A la barra inferior es troben els botons que s'encarreguen del control de l'execució del programa, així com els botons per indicar si el programa es vol simular (al simulador del robot de Polyscope) o executar al robot real. També hi ha indicats els botons per fer *undo*/*redo* dels canvis.

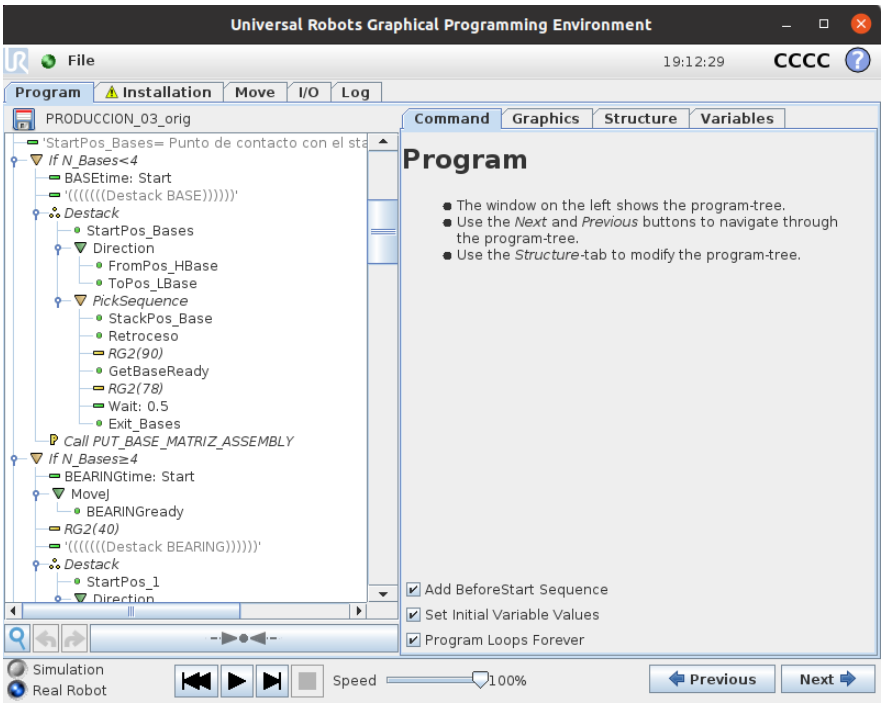

Figura 23: Arbre de programa que va programar Luis Alejandro Chacón Encalada (Font: pròpia)

A la *Figura 23* es pot observar a l'arbre de programa de Polyscope el programa d'en Luis Alejandro Chacón Encalada. La diferència de càrrega visual i cognitiva a l'hora de voler canviar alguna cosa del programa és clara. Per contrapartida, es perd flexibilitat a l'hora de programar.

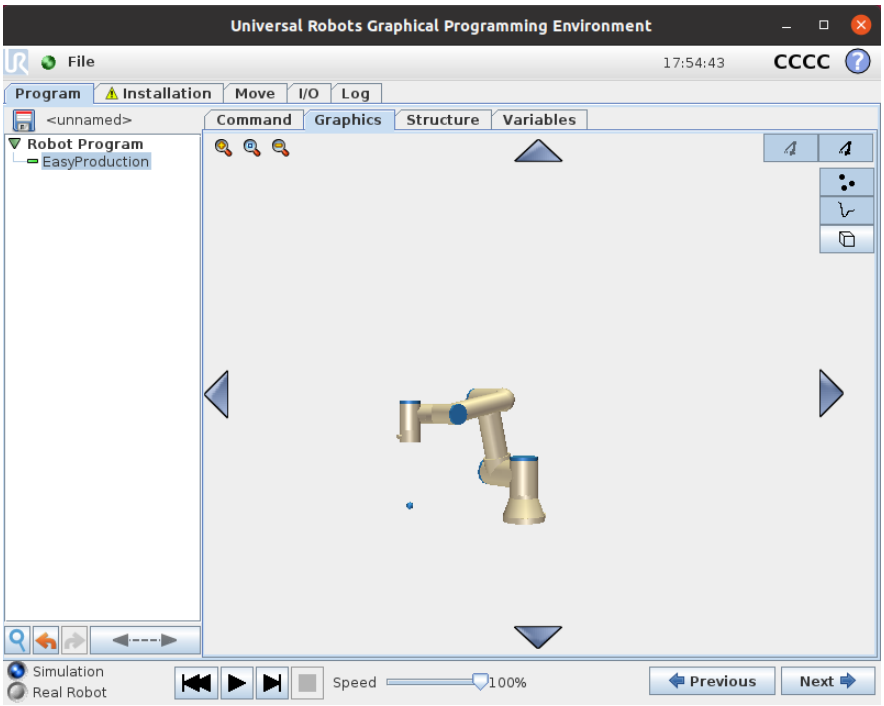

Figura 24: Simulador del robot de Polyscope (Font: pròpia)

El simulador del robot de Polyscope es troba a la pestanya Graphics, com es mostra a la *Figura 24*. Doncs, en cas de voler simular el programa, s'ha d'indicar prement el botó *Simulation* que es troba a la barra inferior a l'esquerra i anar a la pestanya Graphics per veure la simulació.

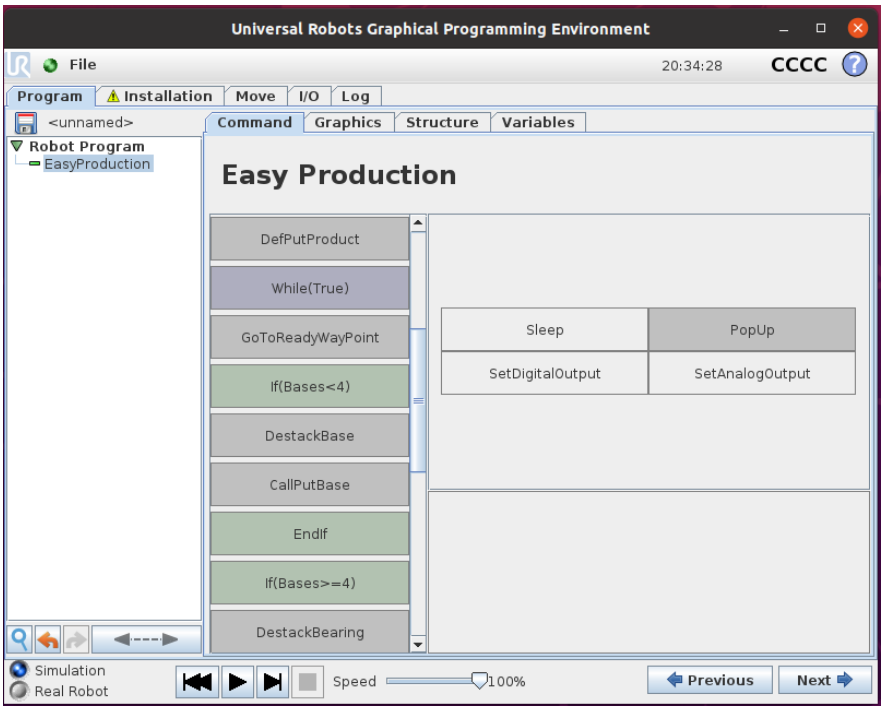

Figura 25: Bloc del panell Opcions seleccionat (Font: pròpia)

Per afegir un dels blocs que es troben al panell **Opcions** al *workflow*, primer s'ha de seleccionar, clicant sobre seu. A la *Figura 25* es mostra un dels blocs del panell **Opcions** seleccionat.

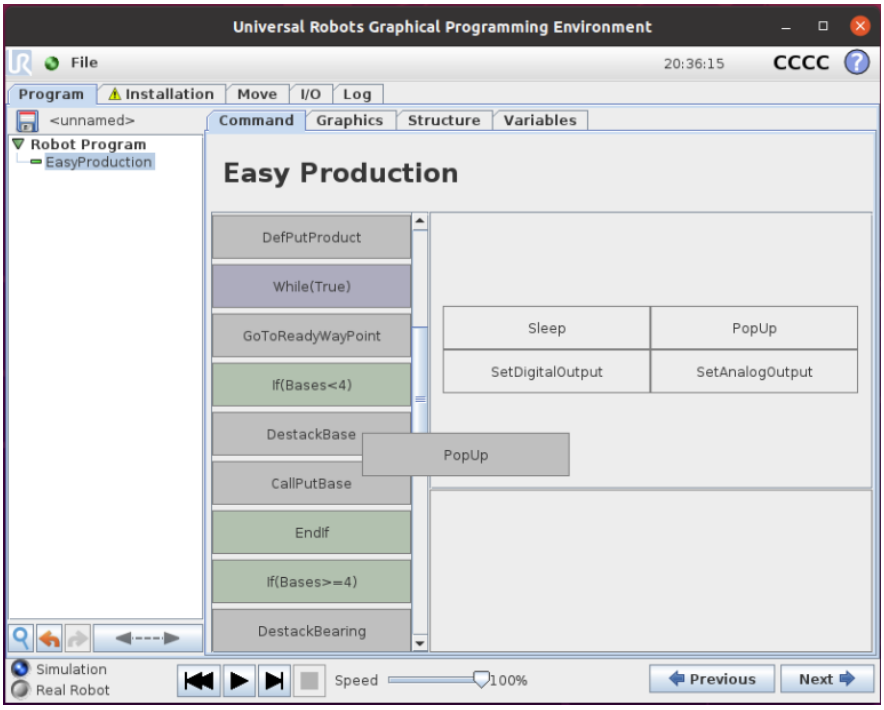

Figura 26: Drag and Drop d'un bloc del panell Opcions (Font: pròpia)

Un cop seleccionat ja es pot fer *drag and drop* del bloc, com es mostra a la *Figura 26*. El bloc s'afegirà sota el bloc del *workflow* on s'hagi fet *drop* (com mostra la *Figura 27*). Si es vol eliminar un dels blocs afegits, al *workflow* només s'ha de fer *drag* d'aquest (arrastrar-lo en qualsevol direcció).

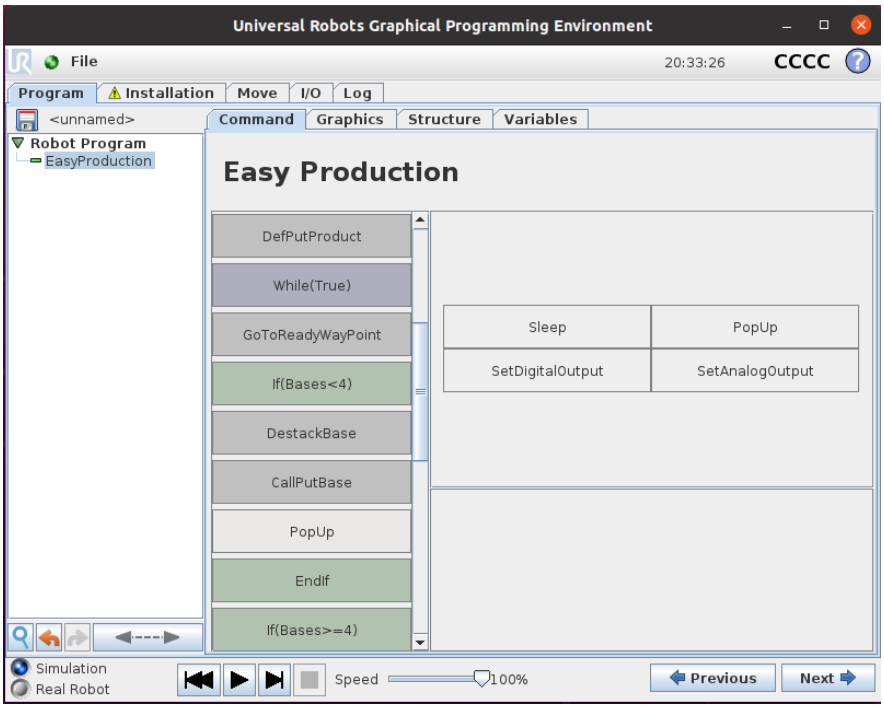

Figura 27: Bloc del panell Opcions afegit al workflow (Font: pròpia)

Per poder visualitzar el panell de paràmetres que té associat cada bloc que es troba al *workflow* s'ha de seleccionar el bloc en qüestió, com es mostra a la *Figura 28*.

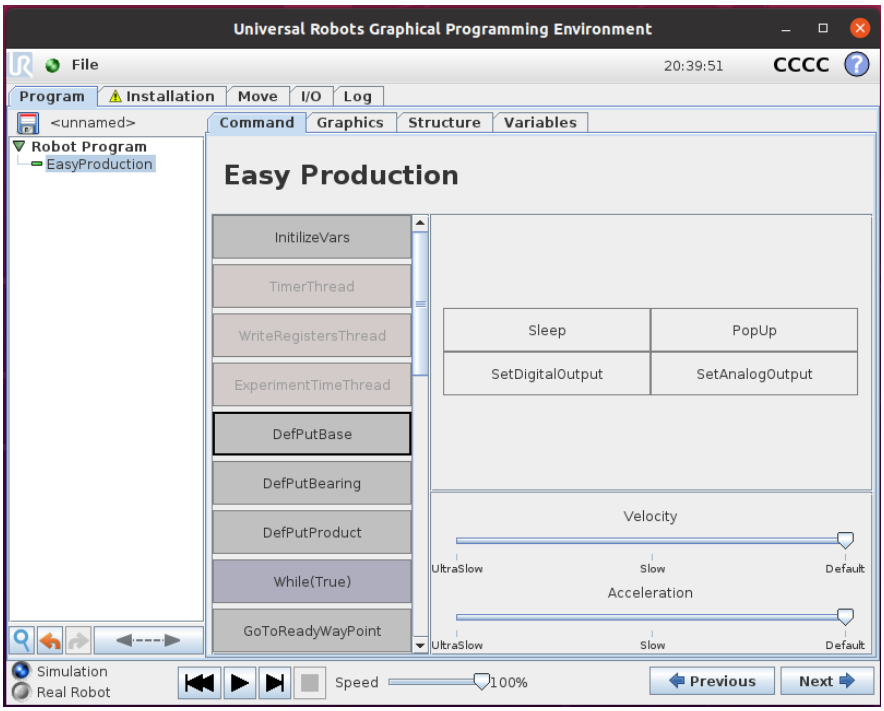

Figura 28: Bloc del workflow seleccionat (Font: pròpia)

A continuació, es mostren els panells de paràmetres dels blocs que es poden parametritzar. N'hi ha que no tenen paràmetres però tenen una descripció informativa sobre l'objectiu del bloc, aquests no es mostraran, doncs, ja s'han descrit els blocs prèviament en aquest document.

Els blocs DefPutBase, DefPutBearing, DefPutProduct, GoToReadyWayPoint, DestackBearing i DespalletizeProduct tenen el panell de paràmetres que es mostra a la *Figura 29*. Aquest permet ajustar la velocitat i l'acceleració del moviment del robot que defineix el bloc quan es dirigeix cap al punt d'aproximació per a realitzar la tasca que té encomanda.

Figura 29: Panell de paràmetres Velocitat/Acceleració (Font: pròpia)

Cal destacar que el bloc DestackBase no té punt d'aproximació perquè és el propi GoToReadyPoint (així ho va programa en Luis Alejandro Chacón Encalada), per tant, no se li pot ajustar ni l'acceleració, ni la velocitat. Com es mostra a la *Figura 30* (exemple d'un bloc que no té paràmetres, però sí un text informatiu).

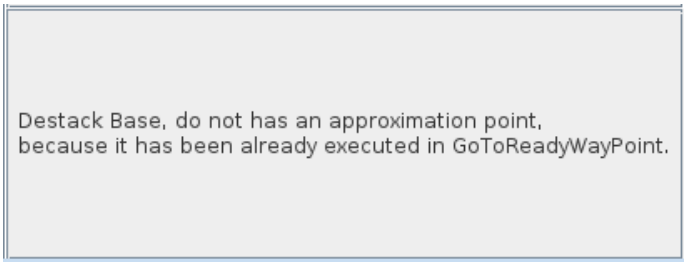

Figura 30: Bloc sense paràmetres amb text informatiu (Font: pròpia)

Els blocs TimerThread, WriteRegistersThread i ExperimentTimeThread consten del panell de paràmetres que es mostra a la *Figura 31*, que permet activar o desactivar el *thread* que s'executa en segon pla a l'execució principal del programa.

Figura 31: Panell de paràmetres que activa o desactiva thread (Font: pròpia)

El bloc Sleep té el panell de paràmetres que es mostra a la *Figura 32*. Permet ajustar quants segons l'usuari vol que es pari l'execució del programa en el rang de 0 a 30 segons.

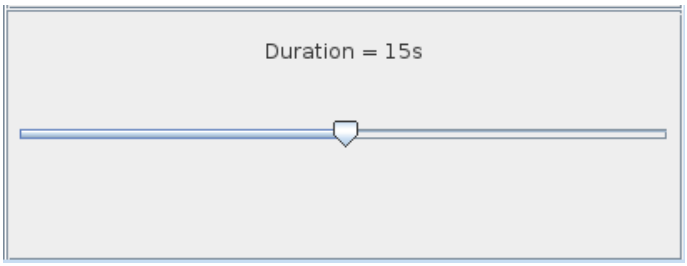

Figura 32: Panell de paràmetres Sleep (Font: pròpia)

El bloc SetDigitalOutput té el panell de paràmetres que es mostra a la *Figura 33*. Permet seleccionar per a quina de les 8 sortides digitals es vol definir el seu valor i si el valor ha de ser *high* o *low*.

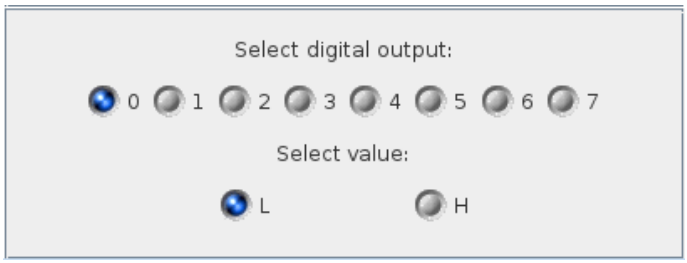

Figura 33: Panell de paràmetres SetDigitalOutput (Font: pròpia)

El bloc SetAnalogOutput té el panell de paràmetres que es mostra a la *Figura 34*. Permet seleccionar per a quina de les 2 sortides analògiques es vol definir el valor i escalar aquest valor mitjançant un *slider* que va del 0 al 1 escalant el valor de la sortida analògica (1 implica una sortida de 10V o 20mA, depenent de com s'hagi configurat la sortida analògica a Polyscope, com es mostra a la *Figura 35*, a la pestanya I/O).

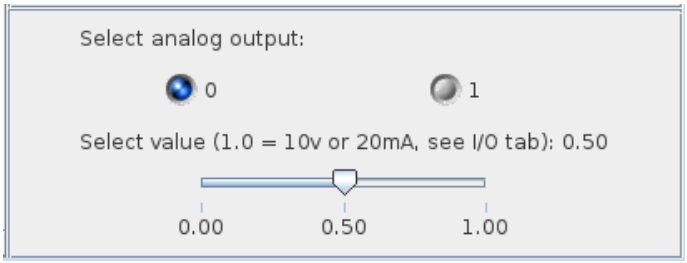

Figura 34: Panell de paràmetres SetAnalogOutput (Font: pròpia)

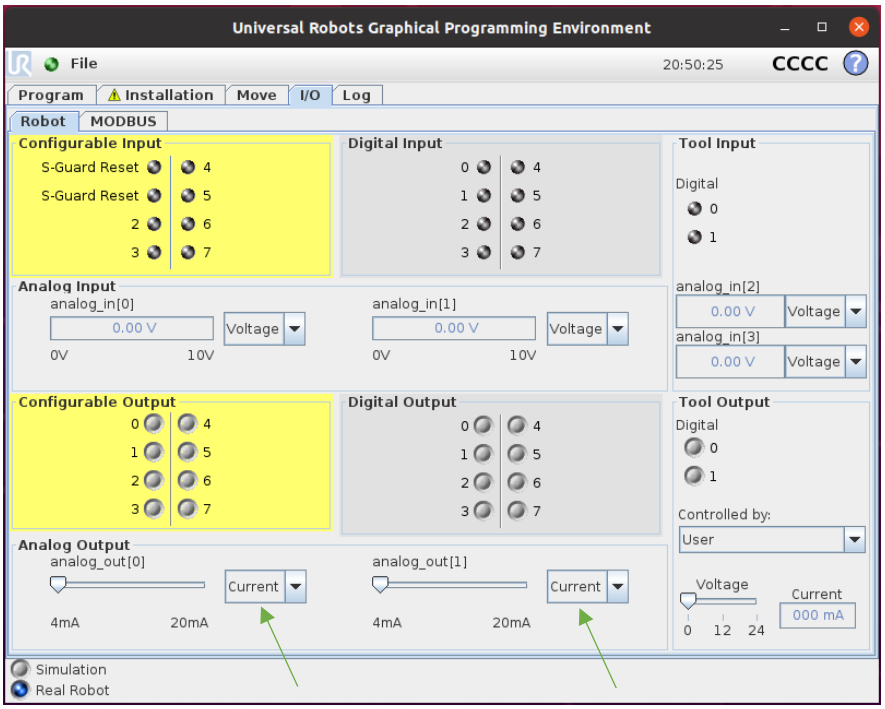

Figura 35: Vista I/O Polyscope (Font: pròpia)

Finalment, el bloc PopUp té el panell de paràmetres que es mostra a la *Figura 36*. Permet definir el títol, el missatge, el tipus de *popup* (missatge, error, avís) i si és bloquejant o no.

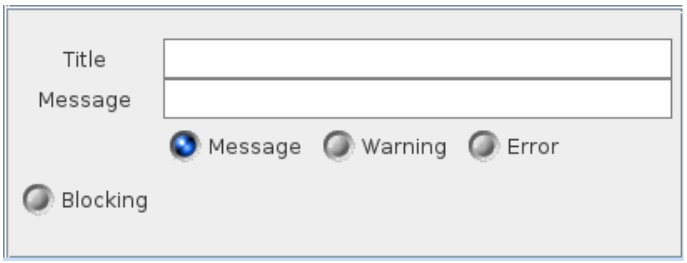

Figura 36: Panell de paràmetres PopUp (Font: pròpia)

3.7. Problemes

Durant la implementació de la URCap el principal problema ha estat l'eficiència. Doncs, el desenvolupament de la URCap s'ha realitzat majoritàriament mitjançant el simulador del *software* del robot, Polyscope, que s'executa a l'ordinador. Però, el *hardware* de l'ordinador no és el mateix que el *hardware* del robot.

En el simulador de Polyscope tot funcionava a la perfecció, però quan va ser hora de realitzar proves instal·lant la URCap al robot real l'estudiant es va adonar que accions com els *undos/redos*, guardar el programa, canviar de node a l'arbre de programació de Polyscope, tardaven excessivament en fer-se. Primerament, es va suposar que el problema derivava de la funció openView() de la classe EasyProductionProgramNodeContribution, doncs, aquesta funció inicialment utilitzava la llibreria gson per recuperar les instancies dels BlockData a partir dels seus Jsons. Després d'afegir la millora que s'ha esmentat abans d'utilitzar la classe MyStringDeserialization, seguien tardant el mateix, per tant, el problema d'eficiència venia d'un altre lloc.

Després de realitzar algunes proves l'estudiant es va adonar de que el problema era que la interfície gràfica tenia masses components visuals i Polyscope li costava molt "eliminar totes aquestes components" per canviar de vista. El simple fet d'eliminar les imatges de les fletxes entre els blocs del *workflow* va reduir el temps d'executar alguna de les accions esmentades anteriorment en una quarta part aproximadament (en un principi tardaven uns 9 segons amb el *workflow* per defecte, la mateixa prova però sense fletxes uns 1.8 segons).

Les conclusions que es poden treure d'aquests problemes són les següents:

- S'ha de vigilar de no afegir masses components visuals a la interfície gràfica del node de programació.
- Polyscope està més enfocat a interfícies gràfiques estàtiques i no tan dinàmiques.

3.8. Vídeo demostració

Al següent enllaç es pot trobar un vídeo demostració sobre el funcionament de la URCap amb el robot real: https://www.youtube.com/watch?v=7TZ-HXmV-xw

4. Desenvolupament de l'aplicació Android

Aquest apartat resumeix els aspectes més rellevants sobre la implementació i el disseny de l'aplicació Android que s'ha desenvolupat durant aquest treball de fi de grau.

4.1. Introducció

Com ja s'ha esmentat en apartats anteriors, el principal objectiu de l'aplicació Android és monitoritzar l'execució del procés industrial. Per tal objectiu, l'aplicació Android ha de poder comunicar-se amb el robot per rebre les dades necessàries i llavors mostrar-les de la manera més apropiada.

L'aplicació Android desenvolupada permet: mostrar l'estat actual del robot (informació sobre l'estat de les articulacions, l'eina, la placa mare, etc.), controlar a distància l'execució del programa, visualitzar les variables globals del programa que poden representar les KPI (de l'anglès *Key Performance Indicator*), enviar missatges al robot, rebre notificacions quan hi ha una parada d'emergència o de protecció i quan s'executa la comanda URScript popup() al robot. En apartats posteriors s'entrarà amb més detall a cada una d'aquestes funcionalitats implementades.

Pel desenvolupament de l'aplicació Android s'ha utilitzat Android Studio com a IDE (de l'anglès *Integrated Development Environment*). L'aplicació s'ha desenvolupat amb el nivell mínim d'API 16.

Seguint la mateixa línia que la URCap implementada, el nom de l'aplicació és EasyProduction.

4.2. Interfícies de comunicació Universal Robots

Universal Robots ofereix diferents possibilitats a l'hora d'establir interfícies de comunicació amb el robot. Aquest apartat pretén resumir les més rellevants i justificar les que s'han escollit per aquest treball de fi de grau. Aquest apartat és un resum de [34].

Interfícies Primària i Secundària (*Primary/Secondary Interfaces*)

El controlador del robot UR proporciona servidors que envien dades sobre l'estat del robot i que poden rebre comandes URScript. La interfície primària envia dades sobre l'estat del robot i missatges addicionals, mentre que la secundària només envia les dades de l'estat del robot. Aquestes dades són principalment utilitzades per la comunicació entre la GUI i el controlador. Les dos interfícies treballen a una velocitat d'actualització de 10Hz. S'utilitza el protocol TCP/IP.

Interfícies en Temps Real (*Real-time Interfaces*)

La funcionalitat de la interfície en temps real és molt similar a la de les interfícies primària/secundària, envia dades sobre l'estat del robot i pot rebre comandes URScript. La principal diferència és la velocitat d'actualització, que en aquest cas és de 125Hz pels robots CB-Series (500Hz pels robots e-Series). S'utilitza el protocol TCP/IP.

Servidor Dashboard

Un robot de Universal Robots es pot controlar de manera remota enviant comandes simples a la interfície gràfica d'usuari mitjançant una comunicació *socket* TCP/IP. Aquesta interfície s'anomena servidor Dashboard i les seves funcionalitats principals són: carregar, reproduir, pausar i aturar un programa del robot, establir el nivell d'accés de l'usuari i rebre *feedback* sobre l'estat del robot.

Comunicació *socket*

El robot UR també pot establir comunicacions *socket* amb el protocol TCP/IP per comunicar-se amb equips externs. En aquest cas el robot és el client i el dispositiu extern el servidor. El llenguatge URScript té comandes per obrir i tancar comunicacions *sockets*, de la mateixa manera que per rebre i enviar dades.

XML-RPC (*Remot Procedure Call*)

És un mètode de crida d'un procediment remot que utilitza XML per transferir dades entre programes mitjançant *sockets*. Amb aquest mètode el controlador UR pot fer crides a funcions (que poden tenir paràmetres) en un servidor/programa remot i recuperar les dades retornades d'una manera estructurada. Pot ser útil, per exemple, per realitzar càlculs complexes que no estan disponibles a URScript. D'aquesta manera es poden combinar altres paquets de *software* amb URScript.

RTDE (*Real-Time Data Exchange*)

RTDE està pensat per substituir d'una manera robusta la interfície de temps real, esmentada anteriorment. Permet al controlador UR transmetre dades d'una manera més personalitzada. S'utilitza el protocol TCP/IP.

A la *Figura 37* es pot observar un diagrama que resumeix les diferents opcions de comunicació esmentades.

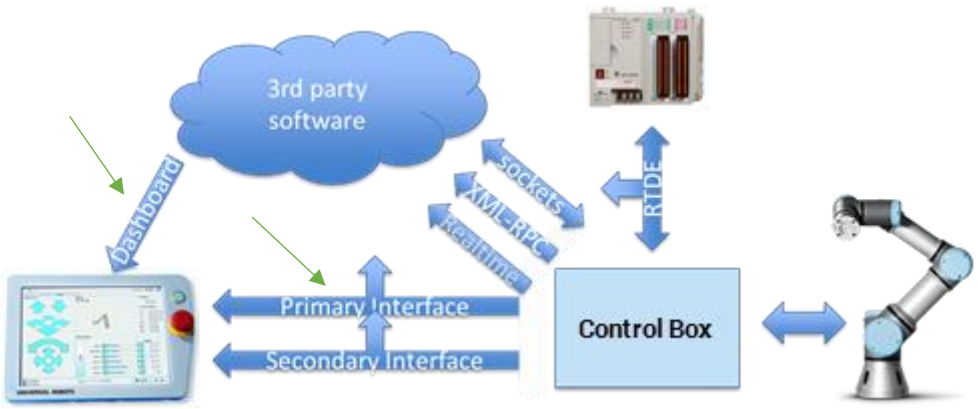

Figura 37: Diagrama interfícies de comunicació UR (Font: [34])

Un cop explicades les diferents opcions de comunicació, quines s'han utilitzat per aquest treball de fi de grau?

- Per tal de rebre les dades corresponents a l'estat del robot (informació sobre l'estat de les articulacions, l'eina, la placa mare, l'estat del programa, l'estat del robot, etc.) s'ha optat per establir una connexió *socket* al port 30001, que és el que utilitza la interfície Primària. S'ha utilitzat la interfície Primària, perquè a banda de la informació de l'estat del robot, també ens interessaven missatges addicionals com els relacionats amb les variables globals i *popups*. Aquesta interfície envia dades a 10Hz, una velocitat més que suficient, doncs, com que són dades que s'han de visualitzar en una interfície gràfica, si s'actualitzen massa ràpid l'ull humà no podria veure gairebé res. D'altra banda, només cal establir la connexió *socket* i ja es comencen a rebre els paquets d'informació, sense haver de configurar res, mentre que altres opcions de comunicació tenen una configuració bastant més complicada.
- Per tal d'enviar comandes simples a Polyscope per poder controlar l'execució del programa o executar *popups* en remot, s'ha optat per utilitzar el servidor Dashboard. És l'opció més senzilla i apropiada per aquest objectiu. Doncs, s'ha establert connexió mitjançant un *socket* connectat al port 29999 que és el corresponent al del servidor Dashboard.

4.3. Paquets descodificats de la interfície Primària

Aquest apartat explica quins dels paquets que el robot envia per la interfície Primària s'han descodificat i quina informació s'ha extret per mostrar-la a l'aplicació Android. A l'annex Especificació dels paquets descodificats, es pot trobar el format complet de cada paquet.

4.3.1. Paquet de l'estat del robot

Aquest paquet conté informació relacionada amb l'estat actual del robot i es divideix en subpaquets com mostra la *Figura 38*.

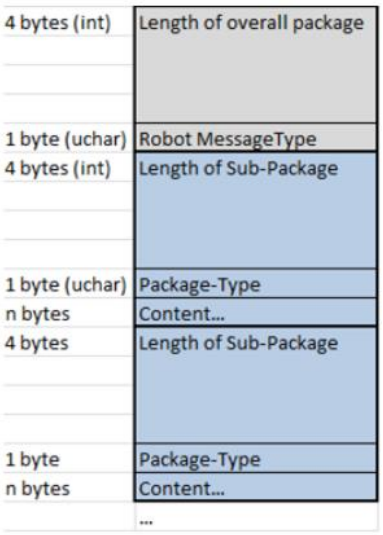

Figura 38: Estructura d'un paquet d'estat del robot (Font: [35])

El primer que s'envia és la llargada del paquet per saber quants bytes s'han de llegir per tenir el paquet complet. Llavors, el tipus del paquet i per cada subpaquet la llargada, el tipus i la informació corresponent. Els subpaquets que es descodifiquen del paquet de l'estat del robot (MESSAGE_TYPE_ROBOT_STATE=16) s'especifiquen a continuació.

Robot mode data

Aquest subpaquet (ROBOT_STATE_PACKAGE_TYPE_ROBOT_MODE_DATA=0) conté dades sobre el mode del robot. Les dades que s'extreuen són:

- bool isEmergengyStopped: booleà que indica si el robot està en parada d'emergència.
- bool isProtectiveStopped: booleà que indica si el robot està en parada de protecció.
- bool isProgramRunning: booleà que indica si el programa carregat al robot s'està executant.
- bool isProgramPaused: booleà que indica si el programa carregat al robot està pausat (si no s'està executant i no està pausat es pot deduir que està parat).
- unsigned char robotMode: byte que indica el mode del robot. Els diferents modes es mostren a la *Figura 39*.

Modes del Robot	
Mode	Descripció
-1	ROBOT_MODE_NO_CONTROLLER
0	ROBOT_MODE_DISCONNECTED
1	ROBOT_MODE_CONFIRM_SAFETY
2	ROBOT_MODE_BOOTING
3	ROBOT_MODE_POWER_OFF
4	ROBOT_MODE_POWER_ON
5	ROBOT_MODE_IDLE
6	ROBOT_MODE_BACKDRIVE
7	ROBOT_MODE_RUNNING
8	ROBOT_MODE_UPDATING_FIRMWARE

Figura 39: Modes del Robot UR (Font: [35])

- unsigned char controlMode: byte que indica el mode de control del robot. Els diferents modes es mostren a la *Figura 40*.

Modes de Control	
Mode	Descripció
0	CONTROL_MODE_POSITION
1	CONTROL_MODE_TEACH
2	CONTROL_MODE_FORCE
3	CONTROL_MODE_TORQUE

Figura 40: Modes de control del Robot UR (Font: [35])

Joint data

Aquest subpaquet (ROBOT_STATE_PACKAGE_TYPE_JOINT_DATA=1) conté dades sobre l'estat de les articulacions del robot. Les dades que s'extreuen per a cada articulació (s'envien en aquest ordre: base (*base*), espatlla (*shoulder*), colze (*elbow*), canell 1 (*wrist 1*), canell 2 (*wrist 2)*, canell 3 (*wrist 3*)) són:

- double q_actual: posició de l'articulació en radians. Per mostrar-ho a l'aplicació es passa a graus.
- float I_actua: intensitat de l'articulació.
- float V_actual: voltatge de l'articulació.
- float T_motor: temperatura del motor de l'articulació.
- uint8_t jointMode: byte que indica el mode de l'articulació. Els diferents modes es mostren a la *Figura 41*.

Modes de les Articulacions	
Mode	Descripció
235	JOINT_MODE_RESET
236	JOINT_MODE_SHUTTING_DOWN
238	JOINT_MODE_BACKDRIVE
239	JOINT_MODE_POWER_OFF
245	JOINT_MODE_NOT_RESPONDING
246	JOINT_MODE_MOTOR_INITIALISATION
247	JOINT_MODE_BOOTING
249	JOINT_MODE_BOOTLOADER
251	JOINT_MODE_VIOLATION
252	JOINT_MODE_FAULT
253	JOINT_MODE_RUNNING
255	JOINT_MODE_IDLE

Figura 41: Modes de les articulacions (Font: [35])

Hi ha modes destinats només per utilització interna que no es contemplen i d'altres que només s'envien a versions de Polyscope 5.x que tampoc.

Tool data

Aquest subpaquet (ROBOT_STATE_PACKAGE_TYPE_TOOL_DATA=2) conté dades sobre l'eina del robot. Les dades que s'extreuen són:

- float toolVoltage48V: voltatge de l'eina (més endavant s'explica un problema relacionat amb aquesta dada).
- float toolCurrent: intensitat de l'eina.
- float toolTemperature: temperatura de l'eina.
- uint8_t toolMode: byte que indica el mode de l'eina. Els diferents modes corresponen a la descripció dels modes de les articulacions especificats a la *Figura 41* (tot i que alguns modes d'articulació no s'envien per l'eina, la documentació que ofereix Universal Robots és ambigua respecte això, així que prefereixo no especificar quins s'envien per l'eina, el que sí que puc assegurar és que els valors que s'envien per l'eina són un subconjunt dels valors que s'han especificat a la *Figura 41* i que les descripcions corresponen a dits valors).

Masterboard data

Aquest subpaquet (ROBOT_STATE_PACKAGE_TYPE_MASTERBOARD_DATA = 3) conté dades sobre la placa mare del robot. Les dades que s'extreuen són:

- float masterBoardTemperature: temperatura de la placa mare.
- float robotVoltage48V: voltatge del robot.
- float robotCurrent: intensitat del robot.

4.3.2.Paquet *popup*

Aquest paquet no conté subpaquets (per tant, s'envia la llargària del paquet, el tipus general, el tipus específic i la informació que conté) i només s'envia a través de la interfície Primària. S'envia quan el robot executa una comanda URScript popup(). El tipus general del paquet és MESSAGE_TYPE_ROBOT_MESSAGE=20. **El tipus específic del paquet que s'especifica a la documentació (ROBOT_MESSAGE_TYPE_POPUP=2) no correspon al que realment es rep (mitjançant prova i error s'ha comprovat que el tipus de paquet és el 9, una mica estrany... Però funciona)**. Les dades que s'extreuen són:

- bool warning: booleà que indica si el *popup* és un avís.
- bool error: booleà que indica si el *popup* és un error. Si no és un avís ni un error es pot deduir que és un missatge normal.
- bool blocking: booleà que indica si el *popup* bloqueja l'execució del programa.
- charArray popupMessageTitle: títol del missatge.
- charArray popupTextMessage: text del missatge.

4.3.3.Paquet noms variables globals

Aquest paquet tampoc conté subpaquets i només s'envia a través de la interfície Primària. El tipus general del paquet és MESSAGE_TYPE_PROGRAM_STATE_MESSAGE=25, l'específic és PROGRAM_STATE_MESSAGE_TYPE_GLOBAL_VARIABLES_SETUP=0. S'envia quan s'inicia l'execució del programa i conté tots els noms de les variables globals (si no hi caben en un sol paquet, s'envien d'altres, són consecutius, aquesta gestió també s'ha tingut en compte). Les dades que s'extreuen són:

- charArray variableNames: noms de les variables globals del programa separades per "\n".

4.3.4.Paquet valors variables globals

Aquest paquet tampoc conté subpaquets i només s'envia a través de la interfície Primària. El tipus general del paquet és MESSAGE_TYPE_PROGRAM_STATE_MESSAGE=25, l'específic és PROGRAM_STATE_MESSAGE_TYPE_GLOBAL_VARIABLES_UPDATE=1. S'envia cada cop que s'actualitzen els valors de les variables globals i conté tots els valors de les variables globals (si no hi caben en un sol paquet, s'envien d'altres, són consecutius, aquesta gestió també s'ha tingut en compte). L'ordre en què s'envia correspon a l'ordre en què s'envien els noms, d'aquesta manera es poden associar correctament. Les dades que s'extreuen són:

- unsigned char variableValues: valors de les variables globals del programa.

Els valors s'envien en una sola tira de bytes que conté per a cada valor:

- 1 byte que indica el tipus de la variable.
- Els bytes de dades necessaris segons el tipus de la variable.
- Acaba amb el byte "\n".

Universal Robots té algunes incoherències en diferents documents sobre els tipus de variables, això es deu a que a les últimes versions s'han anat afegint de nous (per exemple, les matrius) i el valor del byte que indica el tipus de la variable ha anat canviant també. Els possibles tipus de les variables que es contemplen (diria que són totes les possibles per la versió de Polyscope 3.14, però veient que s'estan canviant i la documentació al respecte és ambigua no es pot assegurar que en futures versions l'especificació que es documenta en aquest treball de fi de grau sigui la corresponent) són els que mostra la *Figura 42*.

Tipus	Byte id.	Valor
NONE	0	Buit
CONST_VAR_STRING	3	int16_t valueLength charArray value
VAR_STRING	4	int16_t valueLength charArray value
LIST	17	int16_t listLength For each item: uint8_t valueType DataValueType end
POSE	12	float X float Y float Z float Rx float Ry float Rz
BOOL	13	bool value
NUM	14	float value
INT	15	int32_t value
FLOAT	16	float value
MATRIX	18	int16_t nRows int16_t nColumns For each value in each row: uint8_t valueType DataValueType end

Figura 42: Tipus URScript (Font: [35])

A la *Figura 43* es mostren les mides dels tipus.

Mida dels tipus	
int16_t	2 bytes
int32_t	4 bytes
float	4 bytes
double	8 bytes
bool	1 byte
uint8_t	1 byte

Figura 43: Mida dels tipus (Font: [35])

Cal destacar que experimentalment s'ha comprovat que les llistes poden contenir valors dels tipus: NONE, POSE, BOOL, NUM, FLOAT, INT. Mentre que les matrius poden contenir valors dels tipus: NONE, NUM, FLOAT, INT. Com s'ha especificat abans, possiblement en futures actualitzacions s'afegeixen més tipus.

4.4. Comandes Dashboard utilitzades

Les comandes de la interfície Dashboard que s'han utilitzat són les següents:

- play: inicia l'execució del programa.
- pause: pausa l'execució del programa.
- stop: para l'execució del programa.
- popup: llança un *popup* tipus missatge al robot.

4.5. Aplicació Android

Aquest apartat defineix els conceptes Android que estan implicats a l'aplicació desenvolupada i explica com s'han utilitzat per assolir els objectius del treball de fi de grau.

4.5.1. Activitats Android

La classe Activity és un component molt important quan es desenvolupa una aplicació Android. A diferència de les aplicacions que s'inicien amb una crida al mètode main(), les aplicacions Android s'inicien en una instància d'Activity, invocant els mètodes corresponents a les etapes del seu cicle de vida.

La classe Activity és necessària perquè les aplicacions mòbils, a diferencia de les versions per escriptori, no s'inicien sempre de la mateixa manera. Poden iniciar-se des d'una notificació i mostrar-te una vista relacionada amb aquesta, poden iniciar-se des d'altres aplicacions i mostrar-te una altra vista (per exemple, quan vols compartir una imatge, inicies una aplicació de xarxes socials i aquesta mostra directament la vista de contactes), etc. Doncs, la interacció amb l'usuari no sempre comença des del mateix lloc.

La classe Activity està dissenyada per facilitar aquest paradigma. Quan, per exemple, una aplicació invoca una altra, l'aplicació que realitza la crida, invoca una Activitat de l'altra aplicació, no l'aplicació en si. En resum, una activitat és el punt d'entrada per la interacció d'una aplicació amb l'usuari.

Les activitats implementen vistes (pantalles) d'una aplicació. És a dir, una activitat proporciona la finestra on l'aplicació dibuixa la interfície gràfica d'usuari. Una activitat s'implementa com a subclasse de la classe Activity. S'anomena activitat principal, la que s'obra quan l'aplicació s'inicia prement la seva icona.

Totes les activitats s'executen al UI *thread*, és molt important no realitzar operacions bloquejants (per exemple, connexions *socket*) en aquest *thread*, doncs, hi hauria la possibilitat de bloquejar la GUI de l'aplicació derivant a un ANR (de l'anglès *Application Not Responding*).

Per tal d'assolir els objectius que presenta aquest treball de fi de grau s'han implementat vàries activitats (més endavant es defineixen les activitats amb més detall):

- MainActivity: activitat principal.
- AboutActivity: activitat que mostra algunes dades de referència de l'aplicació.
- RobotStateActivity: activitat on es mostra l'estat del robot i es pot controlar l'execució del programa.
- GuideActivity: activitat que mostra una guia per poder entendre la informació que mostra l'activitat RobotStateActivity.
- GlobalVariablesActivity: activitat on es mostren les variables globals del programa. L'usuari les pot filtrar per veure només les que li interessen.
- MessageActivity: activitat que mostra una entrada de text on l'usuari pot escriure un missatge per enviar-li al robot com a *popup*.

Aquest apartat conté informació extreta de [36].

4.5.2.Serveis Android

Un servei Android (*Service*) és una component d'una aplicació que permet executar operacions de llarga duració en segon pla. A diferència de les activitats, un servei no proporciona una interfície gràfica. Un cop s'inicia el servei, aquest pot seguir executant-se inclús si l'usuari canvia d'aplicació o la tanca. Per exemple, un servei pot gestionar transaccions de xarxa, reproduir música, realitzar operacions d'entrada/sortida de fitxers, etc.

Les activitats poden lligar-se (*bind*) a un servei per poder adquirir informació sobre aquest i mostrar-la a la interfície gràfica. Podem destacar dos tipus de serveis:

- *Started Service*: aquest tipus de servei s'inicia amb la crida de la funció startService() i no para d'executar-se fins a cridar stopService() o stopSelf() (quan el servei gestiona la seva pròpia parada).
- *Bound Service*: aquest tipus de servei s'inicia quan alguna de les activitats es lliga (*bind*) a aquest. Mentre hi hagi activitats lligades al servei, aquest no es destrueix. Quan no té cap activitat lligada (les activitats que estaven lligades fan *unbind*) es destrueix.

A la *Figura 44* podem veure el cicle de vida dels dos tipus de serveis descrits.

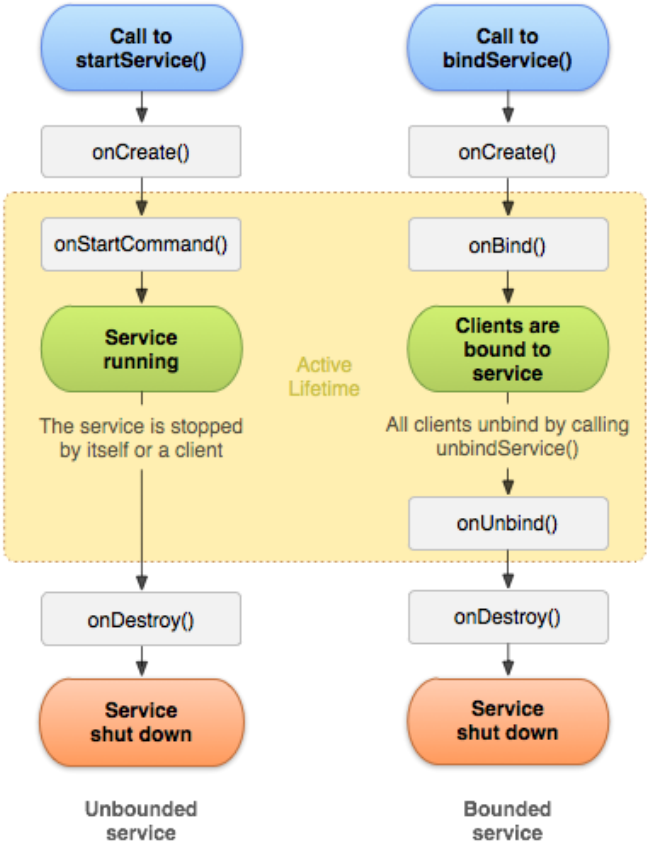

Figura 44: Tipus de serveis Android (Font: [37])

D'altra banda, un servei pot ser:

- *Background Service*: l'usuari del dispositiu mòbil no té cap *feedback* que el notifiqui que el servei s'està executant.
- *Foreground Service*: quan el servei s'està executant mostra una notificació que no es pot eliminar fins que el servei es deixa d'executar. D'aquesta manera, l'usuari del dispositiu mòbil és conscient que hi ha un servei gastant recursos del dispositiu perquè s'està executant.

Per l'aplicació Android desenvolupada en aquest treball de fi de grau s'ha implementat un servei que té com a objectius:

- Establir la connexió *socket* amb la interfície Primària del robot.
- Rebre els paquets d'informació que s'envien i descodificar la seva informació.

Els serveis s'executen al mateix *thread* que les activitats, és a dir, al UI *thread*. Com s'ha mencionat abans, és molt important no realitzar operacions bloquejants en aquest *thread*, doncs, podrien bloquejar el *thread* de la UI de l'aplicació (el *thread* principal) i derivar a un ANR (de l'anglès *Application Not Responding*). Per tant, tota la gestió de la connexió *socket* es fa en un *thread* a part que s'inicia des del servei. Com que el *thread* s'inicia des del servei, no corre el risc de que el sistema operatiu el mati quan cap activitat de l'aplicació és visible o es tanca l'aplicació.

Però, quin tipus de servei s'ha implementat?

- A partir del moment que l'aplicació es connecta amb el robot, el servei s'ha d'estar executant indefinidament fins que l'usuari decideixi desconnectar-se (encara que no hi hagi cap activitat lligada al servei). **(*Started Service*)**
- Les activitats s'han de poder lligar (*bind*) al servei per poder accedir a la informació que conté i mostrar-la a les interfícies gràfiques corresponents. **(*Bound Service*)**
- Per tant, s'ha implementat un servei híbrid. El servei s'inicia com un *Started Service*, així doncs, s'executa indefinidament fins que es crida a stopService() o stopSelf(), independentment de si té activitats lligades o no. Mentre el servei s'està executant les activitats poden lligar-se (*bind*) a aquest per poder accedir a la informació que conté. Aquest plantejament de servei és totalment vàlid ja que es contempla a la documentació oficial d'Android [38].
- D'altra banda, les aplicacions orientades al nivell API 26 o superiors, tenen certes restriccions [39] dirigides als *background services* (serveis en segon pla) quan l'aplicació no es troba en primer pla (el servei en segon pla s'acaba aturant al cap d'un temps). Això és degut a que Android vol reduir el màxim possible l'ús dels recursos limitats dels dispositius mòbils, doncs, si l'usuari no sap que un servei s'està executant en segon pla aquest acabarà sent eliminat pel sistema operatiu. Si el servei notifica a l'usuari que s'està executant (*foreground service* o servei en primer pla), el sistema operatiu no el mata, perquè l'usuari és conscient de que està invertint recursos del seu dispositiu per l'execució del servei en qüestió. Si pensem en aplicacions que executen serveis com reproductors de música o compta passos, aquests sempre mostren una notificació perquè l'usuari en sigui conscient i evitar que el sistema operatiu acabi matant al servei.
- Per tant, el servei implementat llança una notificació, és a dir, és un servei en primer pla (*foreground service*). El servei ha d'executar-se indefinidament fins que l'usuari decideixi desconnectar-se del robot, independentment de si l'aplicació està en primer pla

o no (si és visible o no) o si l'aplicació esta tancada o no. D'aquesta manera, el servei estarà descodificant paquets d'informació constantment i, tot i que no s'estigui mostrant la informació descodificada a cap interfície gràfica (en el cas de que l'aplicació no sigui visible en aquell moment), podrà enviar notificacions si s'escau (si rep algun tipus d'informació que s'ha de notificar).

El servei està implementat a la classe NetworkService que és subclasse de Service. NetworkService té com atribut una instància de la classe TcpIpConnection que gestiona tota la comunicació i descodificació de la informació que envia el robot a través de la interfície Primària. Cal recordar que tota aquesta gestió es du a terme en un *thread* a part que llança el servei.

TcpIpConnection té com atributs una instància de:

- RobotModeData: subclasse de SubPackage que s'encarrega de descodificar i guardar la informació corresponent als paquets de dades del mode del robot.
- JointData: subclasse de SubPackage que s'encarrega de descodificar i guardar la informació corresponent als paquets de dades de les articulacions del robot.
- ToolData: subclasse de SubPackage que s'encarrega de descodificar i guardar la informació corresponent als paquets de dades de l'eina del robot.
- MasterBoardData: subclasse de SubPackage que s'encarrega de descodificar i guardar la informació corresponent als paquets de dades de la placa mare del robot.
- GVarsData: s'encarrega de descodificar i guardar la informació corresponent als paquets de dades de les variables globals del programa que s'està executant.
- PopUpData: s'encarrega de descodificar i guardar la informació corresponent als paquets de dades que s'envien si s'executa una comanda URScript popup().

Les classes que són subclasse de SubPackage, són classes que gestionen paquets que són subpaquets d'un altre paquet.

Per tant, depenent del tipus de paquet de dades que rep TcpIpConnection, aquest crida als mètodes pertinents de la instància corresponent al paquet perquè el descodifiqui i guardi la informació descodificada.

El servei té *getters* que retornen les instàncies de les classes que contenen la informació dels paquets descodificats. D'aquesta manera, si una activitat necessita aquesta informació per mostrar-la, es lliga (*bind*) al servei i té accés als *getters* per poder obtenir les instàncies i, per tant, la informació.

Cal remarcar que la connexió Dashboard no es realitza al servei. Això es deu a que no és necessari mantenir aquesta connexió tota l'estona, només es necessita quan l'aplicació envia una comanda al robot, per tant aquesta comunicació la gestiona l'activitat corresponent en un *thread* a part (per exemple, quan l'activitat detecta que l'usuari vol parar l'execució del programa, és a dir, ha premut el botó parar, aquesta llança un *thread* que es connecta via *socket* a Dashboard, envia la comanda corresponent i llavors tanca la connexió). La classe DashBoardConnection gestiona aquesta connexió.

Aquest apartat conté informació extreta de [37].

4.5.3.Notificacions Android

Com ja s'ha esmentat a l'apartat anterior, el servei és l'encarregat d'enviar notificacions al dispositiu mòbil, d'aquesta manera, tot i que l'aplicació estigui tancada o en segon pla, l'usuari pot seguir rebent notificacions. Les notificacions que es gestionen són les següents:

- Quan el robot realitza una parada d'emergència. Es llança una notificació indicant que hi ha hagut una parada d'emergència. Si l'usuari elimina la notificació i la parada d'emergència continua, es torna a notificar, fins que la parada d'emergència es desactivi.
- Quan el robot realitza una parada de protecció. Es llança una notificació indicant que hi ha hagut una parada de protecció. Si l'usuari elimina la notificació i la parada de protecció continua, es torna a notificar, fins que la parada de protecció es desactivi.
- Quan el robot executa la comanda URScript popup(). En aquest cas la notificació mostra el títol del *popup*, el missatge, el tipus de *popup* (missatge, avís, error) i si és bloquejant o no.

4.6. Disseny de les vistes

Aquest apartat pretén mostrar el disseny de cada vista (pantalla) de l'aplicació Android desenvolupada.

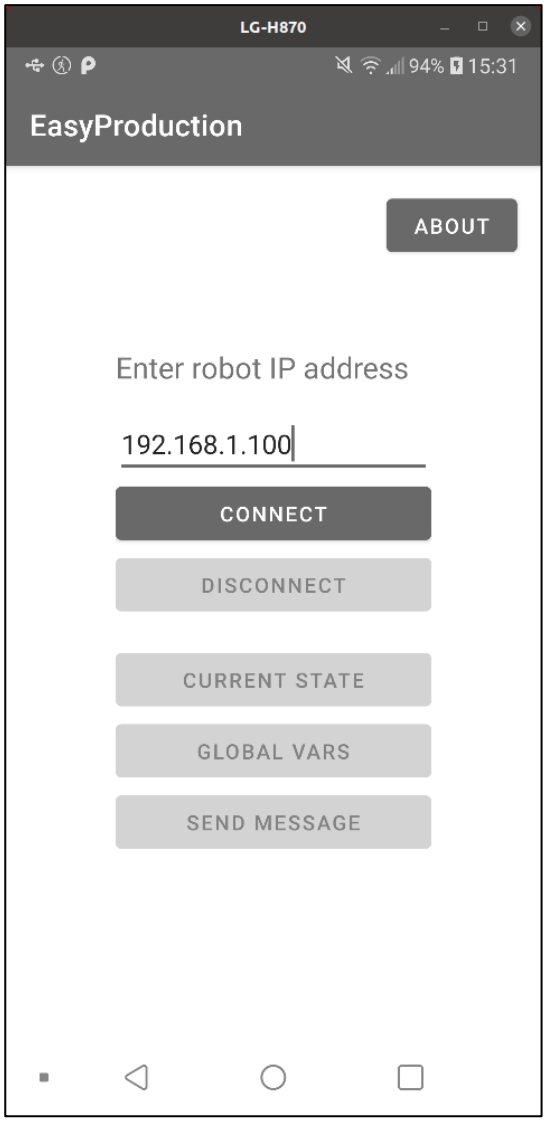

Figura 45: MainActivity (Font: pròpia)

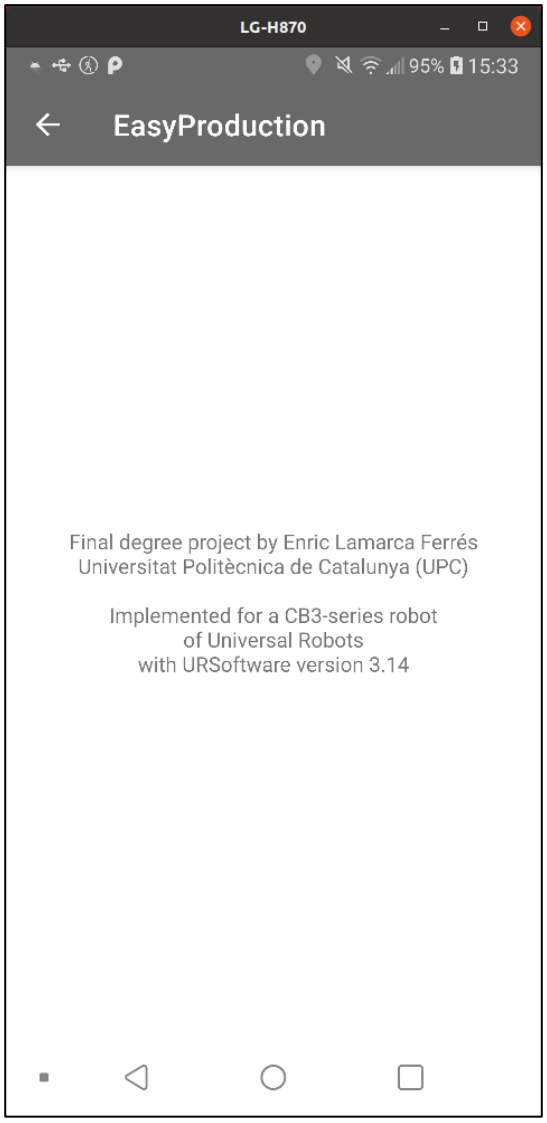

Figura 46: AboutActivity (Font: pròpia)

La pantalla principal (MainActivity) és la que mostra la *Figura 45*. Té una entrada de text on s'ha d'introduir la IP del robot per poder-s'hi connectar mitjançant el botó CONNECT. Per desconnectar-se s'ha de prémer el botó DISCONNECT (habilitat quan hi ha la connexió establerta). Un cop l'aplicació es connecta amb el robot, el servei s'inicia i es llança la notificació (servei en primer pla).

El botó ABOUT (habilitat tot i no haver la connexió establerta) mostra l'activitat AboutActivity. L'activitat AboutActivity és la que mostra la *Figura 46*. Conté informació de referència sobre l'aplicació.

La MainActivity també té una sèrie de botons que es mantenen deshabilitats fins que l'aplicació no es connecta amb el robot. El botó CURRENT STATE mostra l'activitat RobotStateActivity. El botó GLOBAL VARS mostra l'activitat GlobalVariablesActivity. El botó SEND MESSAGE mostra l'activitat MessageActivity.

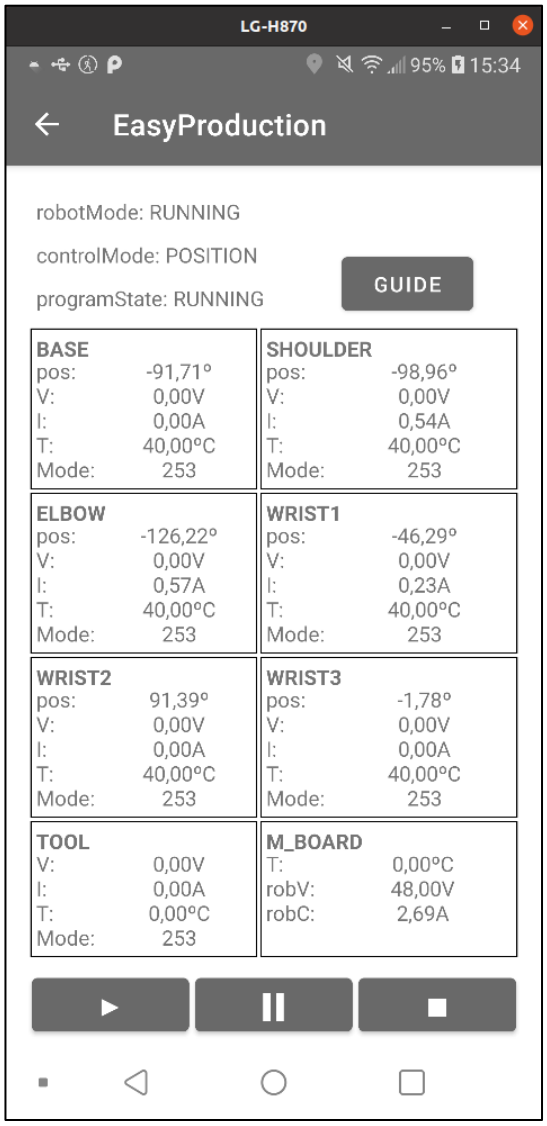

Figura 47: RobotStateActivity (Font: pròpia)

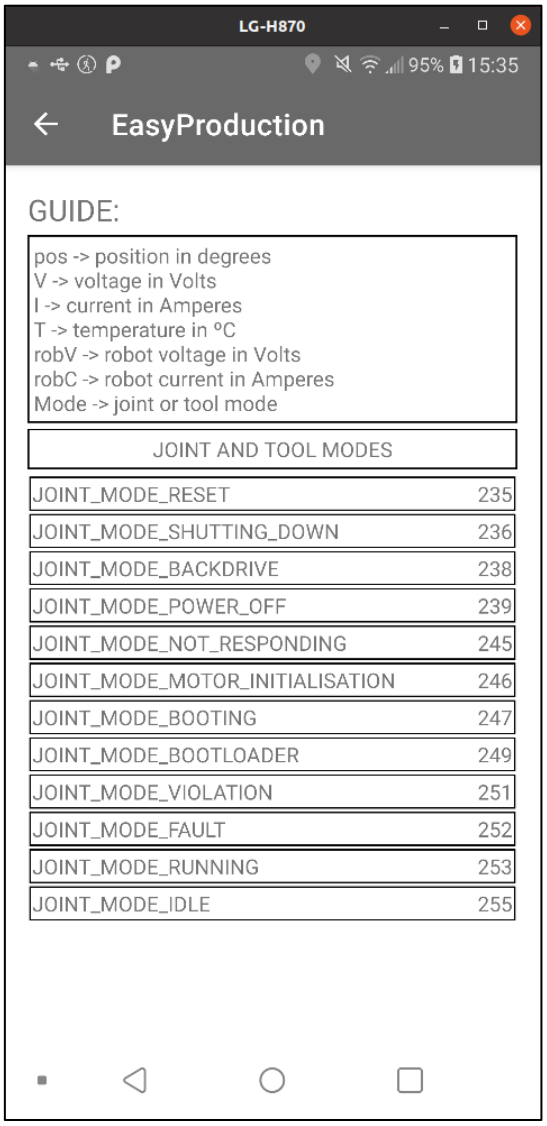

Figura 48: GuideActivity (Font: pròpia)

La *Figura 47* mostra l'activitat RobotStateActivity. Aquesta activitat conté la informació del mode del robot, del mode de control del robot, l'estat del programa, la informació de les articulacions, de l'eina i de la placa mare. Quan hi ha una parada d'emergència o de protecció, a part de la notificació, surt un missatge informatiu en vermell a la cantonada de dalt a la dreta. La informació s'actualitza cada 0,5 segons per no saturar el UI *thread*.

A la part inferior té tres botons: el *play* que inicia l'execució del programa, el *pause* que pausa l'execució del programa i el *stop* que para l'execució del programa.

(*Cal destacar que molts dels valors surten a 0 perquè per fer aquestes captures l'aplicació estava connectada al simulador de Polyscope instal·lat a l'ordinador de l'estudiant, doncs, el simulador no té cap robot real connectat*)

El botó GUIDE mostra l'activitat GuideActivity que conté una guia informativa sobre com interpretar la informació de RobotStateActivity. La *Figura 48* mostra aquesta activitat.

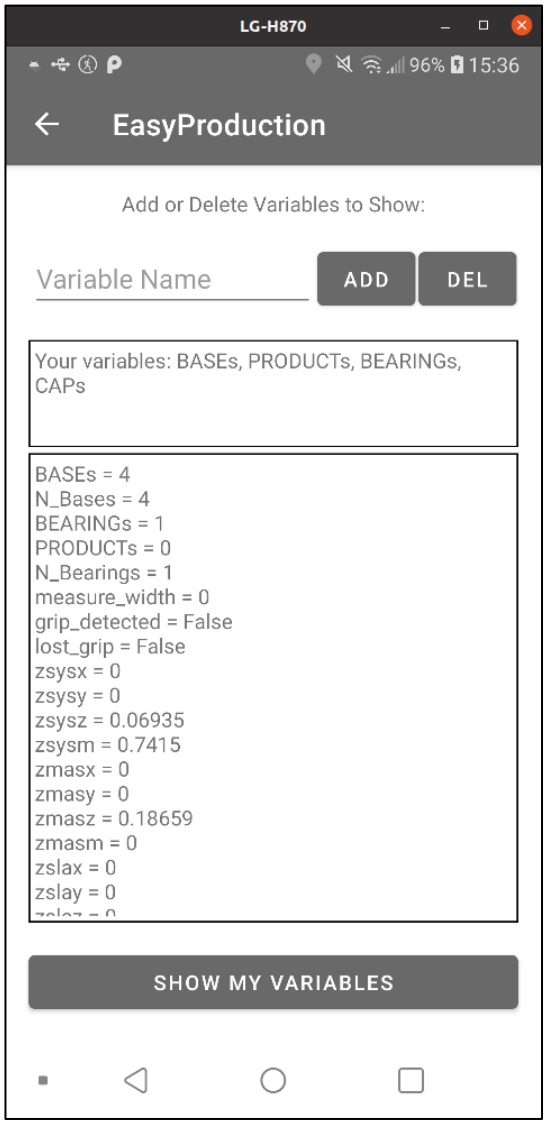

Figura 49: GlobalVariablesActivity amb totes les variables (Font: pròpia)

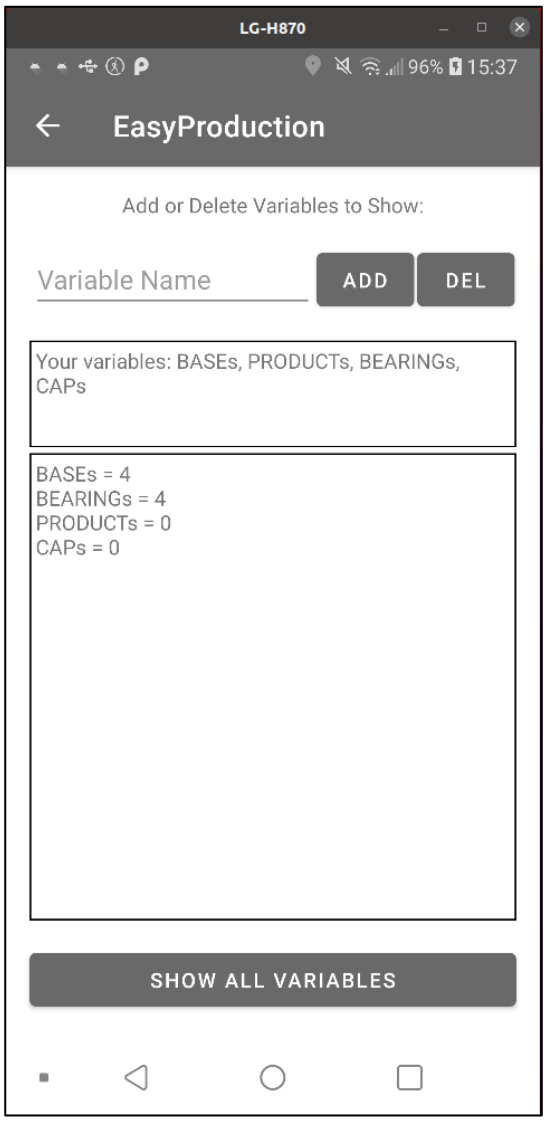

Figura 50: GlobalVariablesActivity amb les variables de l'usuari (Font: pròpia)

La *Figura 49* mostra l'activitat GlobalVariablesActivity amb totes les variables globals del programa, mentre que la *Figura 50* mostra només les introduïdes per l'usuari. L'usuari pot afegir les variables que vol veure escrivint el seu nom a l'entrada de text i prement el botó ADD. Si vol eliminar alguna de les que ha entrat, ha d'introduir el nom de la variable a eliminar i prémer el botó DEL. Els noms de les variables introduïdes per l'usuari es guarden a la memòria del dispositiu mòbil, d'aquesta manera l'usuari no les ha d'estar introduint cada cop que obra l'aplicació.

El botó de la part inferior té una funcionalitat *toggle*. Quan l'activitat està mostrant totes les variables, el botó inferior és SHOW MY VARIABLES, prement-lo, l'activitat només mostra les variables que l'usuari ha introduït. Quan l'activitat mostra només les variables que ha introduït l'usuari, el botó inferior és SHOW ALL VARIABLES, prement-lo, l'activitat mostra totes les variables del programa.

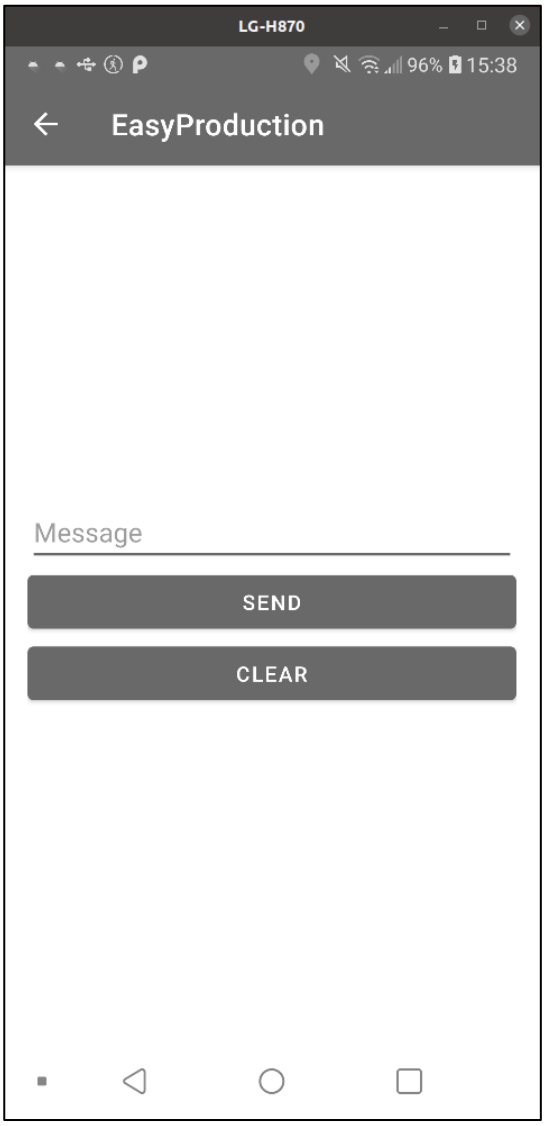

Figura 55: MessageActivity (Font: pròpia)

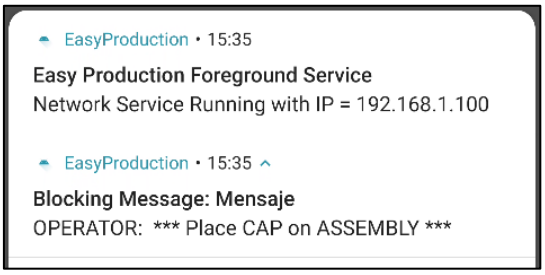

Figura 54: Notificació servei i notificació popup (Font: pròpia)

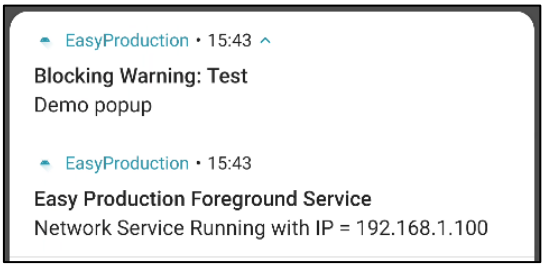

Figura 53: Notificació servei i notificació warning popup (Font: pròpia)

Figura 52: Notificació parada d'emergència (Font: pròpia)

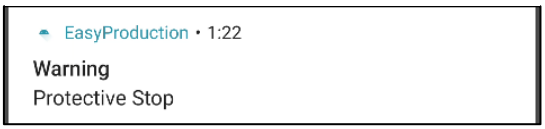

Figura 51: Notificació parada de protecció (Font: pròpia)

La *Figura 55* mostra l'activitat MessageActivity. Aquesta té una entrada de text on l'usuari pot escriure el text del missatge que es vol enviar al robot com a *popup*. El missatge s'envia prement el botó SEND. Per netejar l'entrada de text s'ha de prémer el botó CLEAR.

La *Figura 54* a part de la notificació del servei, mostra una notificació que s'ha enviat degut a que el robot ha executat una comanda URScript popup(). Aquesta notificació conté el tipus de *popup* (missatge, avís, error, bloquejant o no), el títol del *popup*, en aquest cas "Mensaje" i el text del *popup* "OPERATOR: ***Place CAP on ASSEMBLY***". La *Figura 53* mostra un altre exemple d'aquest tipus de notificació, però en aquest cas el *popup* és un avís bloquejant amb títol "Test" i text "Demo popup".

Les notificacions de parada d'emergència i de protecció es mostren a *Figura 52* i a *Figura 51*, respectivament.

4.7. Problemes

L'únic problema que s'ha detectat és el valor del voltatge de l'eina. Quan l'aplicació es connecta al robot real, aquest envia un voltatge de l'eina d'uns 450,0V (un voltatge massa elevat que no té sentit), mentre que la intensitat i la temperatura concorden i tenen valors lògics. Quan es descodifiquen aquest valors van "un darrere l'altre", així que és estrany que uns surtin amb valors que concorden i el voltatge no... Com que l'eina és de tercers (no és de Universal Robots), pot ser que enviï una lectura errònia del voltatge.

D'altra banda, si el servei detecta que ha perdut la connexió amb el robot, aquest tanca la connexió. De totes maneres, poden saltar excepcions, les quals provoquen que el comportament de l'aplicació no sigui l'esperat, si és el cas, el que es recomana fer és desconnectar i tornar a connectar. El que sí s'assegura és que si en un principi s'intenta connectar a una IP incorrecta o que no està disponible en aquell moment, l'aplicació no s'hi connectarà, el problema està en si un cop connectada perd la connexió.

5. Definició de l'ontologia

Aquest apartat explica com s'ha definit l'ontologia que es presenta en aquest treball de fi de grau. El seu objectiu principal és dotar de semàntica al conjunt d'operacions, funcionalitats i macro-moviments que gestionen els productes desenvolupats.

5.1. Introducció

Quan es defineix una ontologia es necessiten fer vàries iteracions al procés de desenvolupament per anar perfilant-la depenent de les necessitats del projecte. Degut a la falta de temps, l'ontologia que presenta aquest treball de fi de grau és una primera versió (primera iteració) de l'ontologia que dota de semàntica els productes desenvolupats. En futurs projectes l'ontologia podria escalar més, definir més restriccions, relacions, classes, etc. Doncs, l'ontologia presentada és una base que es pot ampliar i definir amb més detall en futurs projectes si s'escau. Pel desenvolupament de l'ontologia s'ha utilitzat Protégé 5.5. S'ha guardat amb el format OWL/XML syntax.

5.2. Conceptualització

Aquest apartat pretén donar una descripció dels conceptes del domini objectiu per aquesta ontologia.

Conceptes generals:

- **Operator**: operador que realitza el treball col·laboratiu amb el robot.
- **Robot**: el robot implicat al procés industrial.
- **Polyscope**: software del robot.
- **MasterBoard**: la placa mare del robot.
- **Joint**: articulació del robot.
- **Tool**: eina del robot.
- **Product**: producte final de la producció industrial.
- **ProductComponent**: part (component) del producte final de la producció industrial.
- **Command**: comanda de control de l'execució del programa (*play*, *pause*, *stop*).

Conceptes de la URCap (node de programació):

- **ProgramNode**: node de programació que ofereix la URCap implementada.
- **Block**: bloc que representa un macro-moviment o una part del procés industrial.
- **Workflow**: conjunt de blocs que representen el flux del procés industrial.

Conceptes de l'aplicació Android:

- **App**: aplicació Android.
- **View**: vista (pantalla, activitat) de l'aplicació.
- **Notification**: notificació que llança l'aplicació.

5.3. Formalització: desenvolupament de l'ontologia

Un cop identificats els conceptes bàsics del domini es pot desenvolupar l'ontologia que els representi, mitjançant propietats d'objectes (relacions) o de dades (atributs).

5.3.1. Domini i cobertura de l'ontologia

Com ja s'ha esmentat anteriorment, el domini que ha de cobrir l'ontologia ve definit pels productes que s'han desenvolupat durant aquest treball de fi de grau: la URCap i l'aplicació Android. Doncs, el principal objectiu de l'ontologia és donar semàntica a les funcionalitats dels productes desenvolupats perquè en futurs projectes es pugui afegir control per ordres vocals.

Cal recordar que aquesta ontologia és una primera versió simplificada que intenta representar els conceptes més bàsics del domini objectiu.

5.3.2. Reutilització d'ontologies existents

Actualment, existeixen ontologies que representen processos (*workflows*). Però, reutilitzar una ontologia implica invertir temps de recerca i, un cop trobada, adaptar-la al domini en concret. Doncs, com que el temps ha estat bastant just, s'ha decidit definir-ne una amb els conceptes més bàsics des de zero. Si en un futur es volgués expandir, es podria reutilitzar alguna de les ontologies existents i adaptar-la a l'ontologia presentada en aquest treball de fi de grau (combinar-les).

5.3.3.Definició de les classes i la seva jerarquia

Aquest apartat defineix cada una de les classes de l'ontologia així com els seus atributs i relacions. La *Figura 56* mostra l'estructura de classes dels conceptes més bàsics de l'ontologia.

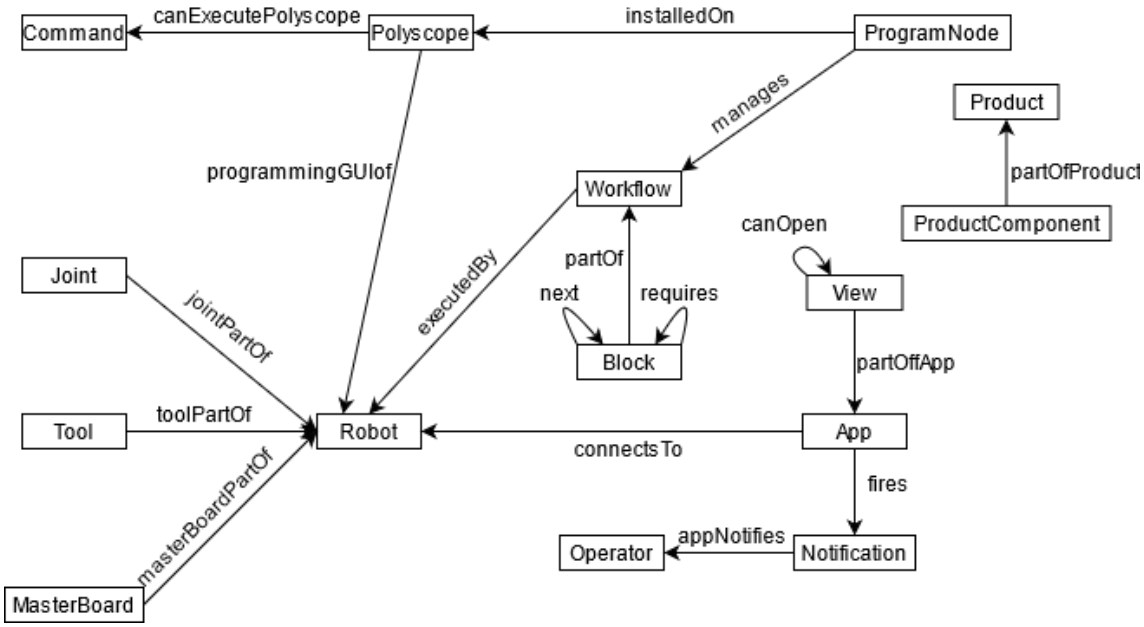

Figura 56: Ontologia amb els conceptes bàsics (Font: pròpia)

App

Aquesta classe representa l'aplicació Android desenvolupada.

Propietats d'Objectes (instàncies):

- connectsTo: instància de la classe Robot que representa el robot amb el qual es connecta l'aplicació Android.
- fires: instància de la classe Notification que representa que l'aplicació llança notificacions.

Propietats de Dades (atributs): -

Block

Aquesta classe representa el concepte general de bloc. És superclasse de classes que representen blocs més específics.

Propietats d'Objectes (instàncies):

- next: instància de la classe Block que representa el bloc següent al bloc actual (ordre *workflow*).

- requires: instància de la classe Block que representa el bloc que requereix el bloc actual. Per exemple, un bloc del tipus *if* requereix un bloc del tipus *endif*.
- partOf: instància de la classe Workflow que representa que el bloc és part del *workflow*. És a dir, el *workflow* (instància de la classe Workflow) està definit per una sèrie de blocs (instàncies de la classe Block).

Propietats de Dades (atributs):

- blockName: string que representa el nom del bloc.
- blockCode: string que representa el codi associat al bloc.

OperationBlock

Aquesta classe representa un bloc que l'usuari pot afegir o eliminar del *workflow*. Aquest tipus de blocs inicialment no es troben al *workflow* del procés industrial però poden ser d'utilitat a l'hora de voler afegir certes operacions al *workflow*. Aquesta classe és superclasse de les classes que representen blocs amb aquestes característiques.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

Popup

Aquesta classe és subclasse de OperationBlock i representa el bloc que té com a codi la comanda URScript popup().

Propietats d'Objectes (instàncies):

- popupNotifies: instància de la classe Operator que indica que el *popup* notifica informació a l'operador.

Propietats de Dades (atributs):

- isError: booleà que indica si el *popup* és un error o no.
- isWarning: booleà que indica si el *popup* és un avís o no.
- messageText: string que conté el text del missatge del *popup*.

BlockingPopup

Aquesta classe és subclasse de la classe Popup i representa un *popup* que és bloquejant, és a dir, que quan es notifica, bloqueja l'execució del programa (procés industrial).

Propietats d'Objectes (instàncies):

- blocksExecutionOf: instància de la classe Workflow que representa el *workflow* que bloqueja el *popup*.

Propietats de Dades (atributs): -

NoBlockingPopup

Aquesta classe és subclasse de la classe Popup i representa un *popup* que no és bloquejant, és a dir, que quan es notifica, l'execució del programa (procés industrial) no es bloqueja.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

SetAnalogOutput

Aquesta classe és subclasse de OperationBlock i representa el bloc que té com a codi la comanda URScript set_standard_analog_out().

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- anOutput: unsignedInt que representa la sortida analògica del robot a la qual es vol definir el seu valor. Com que el robot UR3 CB-series només té dues sortides analògiques, aquest atribut pot tenir com a valor 0 o 1.
- scale: float que representa el nivell d'escalat de la sortida analògica al rang [0, 1] on 1 significa 10V o 20mA (depenent de la configuració de la sortida).

SetDigitalOutput

Aquesta classe és subclasse de OperationBlock i representa el bloc que té com a codi la comanda URScript set_standard_digital_out().

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- digOutput: : unsignedInt que representa la sortida digital del robot a la qual es vol definir el seu valor. Com que el robot UR3 CB-series té vuit sortides digitals, aquest atribut pot tenir com a valor [0, 7].
- outValue: booleà que representa el valor de la sortida digital, 0 significa *low*, 1 significa *high*.

Sleep

Aquesta classe és subclasse de OperationBlock i representa el bloc que té com a codi la comanda URScript sleep().

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- duration: unsignedInt que representa la duració del *sleep*. Pot prendre un valor del rang [0, 30].

ProductionBlock

Aquesta classe representa el conjunt de blocs que formen el procés industrial base. Aquesta classe és superclasse de les classes que representen blocs amb aquestes característiques.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

CallFunc

Aquesta classe és subclasse de la classe ProductionBlock. És superclasse de les classes que representen blocs que fan crides a funcions.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- callFuncInfoLabel: string que representa el text informatiu que mostren aquest tipus de blocs per informar a l'usuari sobre quina funció criden.

CallPutBase

Aquesta classe és subclasse de la classe CallFunc i representa el bloc que fa la crida a la funció encarregada de la col·locació d'una base a la matriu d'assemblatge.

Propietats d'Objectes (instàncies):

- callsBaseFunc: instància de la classe DefPutBase que representa el bloc que té com a codi associat la funció a la qual CallPutBase fa la crida.
- placesBase: instància de la classe Base que representa l'objecte que col·loca el robot després de realitzar la crida a la funció en qüestió.

Propietats de Dades (atributs): -

CallPutBearing

Aquesta classe és subclasse de la classe CallFunc i representa el bloc que fa la crida a la funció encarregada de la col·locació d'un rodament a la matriu d'assemblatge.

Propietats d'Objectes (instàncies):

- callsBearingFunc: instància de la classe DefPutBearing que representa el bloc que té com a codi associat la funció a la qual CallPutBearing fa la crida.
- placesBearing: instància de la classe Bearing que representa l'objecte que col·loca el robot després de realitzar la crida a la funció en qüestió.

Propietats de Dades (atributs): -

CallPutProduct

Aquesta classe és subclasse de la classe CallFunc i representa el bloc que fa la crida a la funció encarregada de la col·locació d'un producte finalitzat a la zona de productes finals.

Propietats d'Objectes (instàncies):

- callsProductFunc: instància de la classe DefPutProduct que representa el bloc que té com a codi associat la funció a la qual CallPutProduct fa la crida.
- placesProduct: instància de la classe Product que representa l'objecte que col·loca el robot després de realitzar la crida a la funció en qüestió.

Propietats de Dades (atributs): -

DefFunc

Aquesta classe és subclasse de la classe ProductionBlock. És superclasse de les classes que representen blocs que defineixen funcions.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- funcAcceleration: float que representa l'acceleració amb la qual el robot fa el moviment cap al punt d'aproximació per realitzar la tasca que defineix la funció.
- funcVelocity: float que representa la velocitat amb la qual el robot fa el moviment cap al punt d'aproximació per realitzar la tasca que defineix la funció.

DefPutBase

Aquesta classe és subclasse de DefFunc i representa el bloc que defineix la funció encarregada de la col·locació d'una base a la matriu d'assemblatge.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

DefPutBearing

Aquesta classe és subclasse de DefFunc i representa el bloc que defineix la funció encarregada de la col·locació d'un rodament a la matriu d'assemblatge.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

DefPutProduct

Aquesta classe és subclasse de DefFunc i representa el bloc que defineix la funció encarregada de la col·locació d'un producte finalitzat a la zona de productes finals.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

FlowInstruction

Aquesta classe és subclasse de ProductionBlock i representa un bloc que defineix el flux d'execució del *workflow*. Per exemple, un *if*, *while*, *endif*, etc.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- flowInstrInfoLabel: string que representa el text informatiu que mostren aquest tipus de blocs per informar a l'usuari sobre la seva finalitat principal.

GetCAPs

Aquesta classe és subclasse de ProductionBlock i representa un bloc encarregat d'avisar a l'operador perquè col·loqui les tapes a la matriu d'assemblatge i d'esperar la seva confirmació per seguir amb l'execució del procés industrial.

Propietats d'Objectes (instàncies):

- needs: instància de la classe Operator que indica que aquest bloc necessita la participació de l'operador que col·labora amb el robot durant l'execució del procés industrial.
- affectsCap: instància de la classe Cap que representa l'objecte que col·loca l'operador.

Propietats de Dades (atributs):

- getCapsInfoLabel: : string que representa el text informatiu que mostra aquest bloc per informar a l'usuari sobre la seva finalitat principal.

GetReadyToPut

Aquesta classe és subclasse de la classe ProductionBlock. És superclasse de les classes que representen blocs que s'encarreguen de preparar l'estat del robot corresponent perquè posteriorment pugui executar alguna de les crides a les funcions que col·loquen objectes (base, rodament, producte).

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

DespalletizeProduct

Aquesta classe és subclasse de GetReadyToPut i representa el bloc que defineix el codi encarregat de treure un producte finalitzat de la matriu d'assemblatge perquè posteriorment es pugui col·locar a la zona de productes finals.

Propietats d'Objectes (instàncies):

- affectsProduct: instància de la classe Product que representa l'objecte el qual es veu afectat durant l'execució d'aquest bloc.

Propietats de Dades (atributs):

- productAcceleration: float que representa l'acceleració amb la qual el robot fa el moviment cap al punt d'aproximació per realitzar la tasca que defineix aquest bloc.
- productVelocity: float que representa la velocitat amb la qual el robot fa el moviment cap al punt d'aproximació per realitzar la tasca que defineix aquets bloc.

DestackBase

Aquesta classe és subclasse de GetReadyToPut i representa el bloc que defineix el codi encarregat de desapilar una base perquè posteriorment es pugui col·locar a la matriu d'assemblatge.

Propietats d'Objectes (instàncies):

- affectsBase: instància de la classe Base que representa l'objecte el qual es veu afectat durant l'execució d'aquest bloc.

Propietats de Dades (atributs):

- destackBaseInfoLabel: string que representa el text informatiu que mostra aquest bloc per informar a l'usuari sobre la seva finalitat principal (en aquest cas, el bloc no executa cap punt d'aproximació, perquè es realitza en un altre bloc, concretament, al GoToReadyWayPoint).

DestackBearing

Aquesta classe és subclasse de GetReadyToPut i representa el bloc que defineix el codi encarregat de desapilar un rodament perquè posteriorment es pugui col·locar a la matriu d'assemblatge.

Propietats d'Objectes (instàncies):

- affectsBearing: instància de la classe Bearing que representa l'objecte el qual es veu afectat durant l'execució d'aquest bloc.

Propietats de Dades (atributs):

- bearingAcceleration: float que representa l'acceleració amb la qual el robot fa el moviment cap al punt d'aproximació per realitzar la tasca que defineix aquest bloc.
- bearingVelocity: float que representa la velocitat amb la qual el robot fa el moviment cap al punt d'aproximació per realitzar la tasca que defineix aquets bloc.

GoToReadyWayPoint

Aquesta classe és subclasse de GetReadyToPut i representa el bloc que defineix el codi encarregat d'executar el primer punt d'aproximació que el robot executa a cada volta del bucle principal que controla el procés industrial. Aquest punt d'aproximació és el punt d'aproximació per a desapilar una base (per aquest motiu al bloc que representa la classe DestackBase no se li pot ajustar la velocitat ni l'acceleració del punt d'aproximació perquè es defineixen en aquest bloc).

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- goReadyAcceleration: float que representa l'acceleració amb la qual el robot fa el moviment cap al punt d'aproximació per realitzar la tasca que defineix aquest bloc.
- goReadyVelocity: float que representa la velocitat amb la qual el robot fa el moviment cap al punt d'aproximació per realitzar la tasca que defineix aquets bloc.

InitializeVars

Aquesta classe és subclasse de la classe ProductionBlock i representa el bloc encarregat d'inicialitzar les variables globals del programa que defineix el procés industrial.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- iniInfoLabel: string que representa el text informatiu que mostra aquest bloc per informar a l'usuari sobre la seva finalitat principal.

Thread

Aquesta classe és subclasse de la classe ProductionBlock i representa un bloc que defineix i inicia un *thread* que s'executa en paral·lel a l'execució del programa del procés industrial.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- activateThread: booleà que indica si el *thread* s'inicia o no (depenent de les necessitats de l'operador).

Command

Aquesta classe és superclasse de les diferents comandes encarregades de controlar l'execució del programa del procés industrial (*play*, *pause*, *stop*).

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

Pause

Aquesta classe és subclasse de la classe Command i representa la comanda *pause* que pausa l'execució del programa del procés industrial.

Propietats d'Objectes (instàncies):

- pauses: instància de la classe Workflow que representa el *workflow* al qual se li envia aquesta comanda.

Propietats de Dades (atributs): -

Play

Aquesta classe és subclasse de la classe Command i representa la comanda *play* que inicia l'execució del programa del procés industrial.

Propietats d'Objectes (instàncies):

- plays: instància de la classe Workflow que representa el *workflow* al qual se li envia aquesta comanda.

Propietats de Dades (atributs): -

Stop

Aquesta classe és subclasse de la classe Command i representa la comanda *stop* que para l'execució del programa del procés industrial.

Propietats d'Objectes (instàncies):

- stops: instància de la classe Workflow que representa el *workflow* al qual se li envia aquesta comanda.

Propietats de Dades (atributs): -

Joint

Aquesta classe representa una articulació del robot.

Propietats d'Objectes (instàncies):

- jointPartOf: instància de la classe Robot que representa que l'articulació és part del robot.

Propietats de Dades (atributs):

- intensity: float que representa la intensitat de l'articulació.
- jointName: string que representa el nom de l'articulació.
- mode: unsignedByte que representa en quin mode es troba l'articulació.
- pos: double que representa en quina posició es troba l'articulació (en graus).

- temperature: float que representa la temperatura de l'articulació.
- voltatge: float que representa el voltatge de l'articulació.

MasterBoard

Aquesta classe representa la placa mare del robot.

Propietats d'Objectes (instàncies):

- masterBoardPartOf: instància de la classe Robot que representa que la placa mare és part del robot.

Propietats de Dades (atributs):

- robIntensity: float que representa la intensitat del robot.
- robVoltage: float que representa el voltatge del robot.
- tempMBoard: float que representa la temperatura de la placa mare.

Notification

Aquesta classe representa una notificació que llança l'aplicació Android.

Propietats d'Objectes (instàncies):

- appNotifies: instància de la classe Operator que representa a l'operador a qui se li notifica la informació corresponent a la notificació.

Propietats de Dades (atributs):

- text: string que representa el text que conté la notificació.
- title: string que representa el títol de la notificació.

Operator

Aquesta classe representa l'operador que realitza el treball col·laboratiu amb el robot.

Propietats d'Objectes (instàncies):

- canAddOrDelete: instància de la classe OperationBlock que representa que l'operador por afegir o eliminar al *workflow* aquest tipus de blocs.

Propietats de Dades (atributs):

- operatorName: string que representa el nom de l'operador.

Polyscope

Aquesta classe representa el *software* del robot anomenat Polyscope.

Propietats d'Objectes (instàncies):

- canExecutePolyscope: instància de la classe Command que indica que Polyscope pot executar comandes.
- programmingGUIof: instància de la classe Robot que indica que Polyscope és la interfície gràfica de programació del robot.

Propietats de Dades (atributs):

- polyscopeVersion: string que representa la versió de Polyscope.

Product

Aquesta classe representa un producte finalitzat.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

ProductComponent

Aquesta classe representa un component del producte final. És superclasse de classes que representen els diferents components del producte final.

Propietats d'Objectes (instàncies):

- partOfProduct: instància de Product que representa que una component de producte és part d'un producte final.

Propietats de Dades (atributs): -

Base

Aquesta classe és subclasse de la classe ProductComponent i representa una base.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

Bearing

Aquesta classe és subclasse de la classe ProductComponent i representa un rodament.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

Cap

Aquesta classe és subclasse de la classe ProductComponent i representa una tapa.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs): -

ProgramNode

Aquesta classe representa el node de programació que s'ha desenvolupat.

Propietats d'Objectes (instàncies):

- installedOn: instància de Polyscope que representa que el node de programació està instal·lat a Polyscope.
- manages: instància de Workflow que indica que el node de programació gestiona la programació del *workflow* de la producció industrial.

Propietats de Dades (atributs): -

Robot

Aquesta classe representa el robot que participa a la producció industrial.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- robotModel: string que representa el model del robot.

Tool

Aquesta classe representa l'eina que té el robot.

Propietats d'Objectes (instàncies):

- toolPartOf: instància de Robot que representa que l'eina és part del robot.

Propietats de Dades (atributs):

- toolIntensity: float que representa la intensitat de l'eina.
- toolMode: unsignedByte que representa el mode de l'eina.
- toolModel: string que representa el model de l'eina.
- toolTemperature: float que representa la temperatura de l'eina.
- toolVoltage: float que representa el voltatge de l'eina.

View

Aquesta classe representa una vista (una activitat) de l'aplicació. És superclasse de classes que representen les vistes que ofereix l'aplicació Android.

Propietats d'Objectes (instàncies):

- canOpen: instància de View que representa que la vista actual pot obrir altres vistes.
- partOfApp: instància de App que representa que la vista forma part de l'aplicació Android.

Propietats de Dades (atributs): -

About

Aquesta classe és subclasse de la classe View i representa la vista *about*.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- aboutText: string que representa el text que conté aquesta vista.

CurrentState

Aquesta classe és subclasse de la classe View i representa la vista que mostra l'estat actual del robot.

Propietats d'Objectes (instàncies):

- canExecute: instància de Command que representa que aquesta vista pot enviar comandes al robot.
- showsInfoAbout: instància de Robot que representa que aquesta vista mostra informació sobre el robot.

Propietats de Dades (atributs): -

GlobalVars

Aquesta classe és subclasse de la classe View i representa la vista que mostra les variables globals del programa del procés industrial.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- allVars: string que representa totes les variables i valors del programa del procés industrial.
- isFiltering: booleà que representa si les variables que es mostren són filtrades (només es mostren les variables que l'usuari vol veure) o no.
- userNameVars: string que representa els noms de les variables que l'usuari vol veure en cas de que la vista estigui filtrant les variables mostrades.

Guide

Aquesta classe és subclasse de la classe View i representa la vista que mostra una guia per saber interpretar els valors que mostra la vista CurrentState.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- guideText: string que representa el text que conté aquesta vista.

Main

Aquesta classe és subclasse de la classe View. Representa la vista principal.

Propietats d'Objectes (instàncies): -

Propietats de Dades (atributs):

- ipToConnect: string que representa la IP del robot amb el qual l'aplicació s'ha de connectar.

SendMessage

Aquesta classe és subclasse de la classe View i representa la vista encarregada d'oferir a l'usuari la possibilitat d'enviar un missatge al robot.

Propietats d'Objectes (instàncies):

- sendMessageAs: instància de Popup que indica que aquesta vista envia un missatge al robot com un *popup*.

Propietats de Dades (atributs):

- messageToSend: string que representa el text del missatge.

Workflow

Aquesta classe representa el *workflow* del procés industrial que està constituït per blocs.

Propietats d'Objectes (instàncies):

- executedBy: instància de Robot que representa que el programa definit pel *workflow* és executat pel robot.

Propietats de Dades (atributs):

- isSimulated: booleà que representa si el *workflow* es vol executar al simulador o al robot real.

---

6. Sostenibilitat

En aquest apartat es valorarà la sostenibilitat del projecte en diferents dimensions: econòmica, ambiental i social.

6.1. Autoavaluació i reflexió

Després de realitzar l'enquesta sobre sostenibilitat presentada pels membres del projecte EDINSOST he pogut extreure certes conclusions.

Al llarg de la meva trajectòria acadèmica, en especial, durant el Grau en Enginyeria Informàtica, m'han explicat en què consisteix la sostenibilitat i el paper tan rellevant que pren a qualsevol tipus de projecte.

Personalment, sempre que penso en un projecte el primer que em ve en ment és l'impacte social que podria tenir. Crec que si no hi ha impacte social, no hi ha projecte, per tant, sempre he prioritzat la sostenibilitat en la dimensió social, per aquest motiu és la que més domino.

D'altra banda, la sostenibilitat ambiental la considero essencial per qualsevol tipus de projecte. Tot i que al sector de la informàtica no s'utilitzen matèries primeres, s'utilitzen molts dispositius tecnològics, els quals sempre s'ha de prioritzar reparar-los, reutilitzar-los i aprofitar al màxim la seva vida útil. Possiblement em falti base teòrica per completar el meu coneixement sobre aquesta dimensió però sóc molt conscient de la importància que té.

Finalment, considero que la dimensió econòmica és la que menys domino i, segurament, és degut a la meva falta d'experiència laboral. Els projectes que he realitzat sempre han estat desenvolupats dins del món acadèmic, on la dimensió econòmica és gairebé inexistent. Ara, realitzant el treball de fi de grau, me n'adono de la rellevància que té la sostenibilitat en la dimensió econòmica i que en certa manera les tres dimensions estan relacionades.

6.2. Dimensió econòmica

Considero que l'estimació del cost total del projecte és bastant realista. S'ha justificat l'estimació del cost/hora dels diferents rols i el Cost de Personal per Activitat. També, s'han tingut en compte costos generals com són amortitzacions, costos energètics i costos derivats de l'espai de treball.

Per tal de donar marge de maniobra en la basant econòmica del projecte, s'ha afegit la partida de contingències, concretament, un 15% del cost CPA + CG. Donat que el percentatge destinat a les partides de contingència de projectes de desenvolupament de *software* es troba entre un 10% i un 20%, crec que el terme intermedi d'un 15% és realista i el més convenient. Els costos dels imprevistos també es justifiquen adequadament.

Tot i que s'ha estimat a l'engròs el nombre d'hores destinades a certes tasques, no suposa un sobrecost, perquè aquestes estimacions a l'alça estan destinades a contrarestar la falta de coneixement en l'àmbit i els problemes tecnològics que segurament sorgiran.

S'ha prioritzat l'ús de programari lliure, per tant, el manteniment del producte desenvolupat no hauria de suposar costos a llarg termini (durant la seva vida útil), pel contrari d'altres possibles solucions al problema presentat (millorar la interacció entre humans i robots industrials) que

necessiten llicència pel seu ús. D'aquesta manera es redueixen els possibles escenaris que podrien perjudicar la viabilitat del projecte.

El cost previst s'ha ajustat bastant al cost final, com s'ha pogut veure en els apartats corresponents a la gestió econòmica.

Facilitar i millorar la interacció entre humans i robots industrials implica estalviar a les empreses inversions econòmiques destinades a la preparació tècnica dels seus empleats. De la mateixa manera, la producció es pot veure afavorida i, per tant, representa un impacte econòmic positiu per l'empresa en general.

6.3. Dimensió ambiental

Des d'un bon principi s'ha intentat reduir l'impacte ambiental negatiu que esdevé per la realització del projecte.

S'ha prioritzat allargar la vida útil de l'ordinador portàtil amb el qual s'implementarà tot el projecte, instal·lant una SSD. Una alternativa hauria sigut comprar-ne un de nou, però l'impacte ambiental hagués estat més negatiu, és preferible reutilitzar els dispositius tecnològics fins a exhaurir la seva vida útil al complet. A més a més, un ordinador portàtil consumeix menys que un de sobretaula.

Pel que fa al robot, si s'espatllés durant el desenvolupament del projecte es prioritzarà la seva reparació o recanvi de peces (preferiblement de segona mà o ja en disposició de la pròpia universitat).

Respecte la documentació, es realitzarà i gestionarà de manera digital. Només s'imprimirà al final per presentar la memòria a la lectura del treball de fi de grau. D'aquesta manera, s'estalvia paper i tinta.

L'impacte ambiental a llarg termini del projecte (vida útil), un cop desenvolupat, està relacionat amb l'impacte econòmic explicat a l'apartat anterior. Es destinaran menys recursos a la preparació del personal i s'agilitzarà la producció industrial gràcies al producte desenvolupat, així doncs, el consum energètic i de recursos com paper, tinta, entre d'altres, es veuran reduïts.

6.4. Dimensió social

A nivell personal, podré endinsar-me al món de la robòtica col·laborativa, una temàtica del meu interès. Aprendré molts conceptes relacionats amb la robòtica i la programació: com programar un robot?, com dissenyar una interfície gràfica intuïtiva?, com definir una semàntica pel conjunt de funcionalitats que ofereixi aquesta interfície?, quina és la millor manera de simular un procés industrial?, com gestionar el conjunt d'operacions i la comunicació ordinador-robot d'una manera eficient?, etc. A part de tots aquests conceptes tècnics, també aprendré a gestionar i organitzar un projecte de recerca i desenvolupament de *software*.

Existeix una necessitat real del projecte, doncs, com ja s'ha introduït anteriorment, la indústria col·laborativa està a l'ordre del dia (Industry 4.0). Cada cop a més sectors és necessària una interacció entre operadors humans i robots que realitzen alguna tasca en concret. Com més propera sigui la interacció humà-robot a una interacció humà-humà, més integrats estaran els

robots als processos col·laboratius, millorant la producció, l'economia, l'ambient de treball, la seguretat, etc.

Finalment, aquest projecte pot servir de base per futurs projectes que puguin afegir control per ordres vocals o altres ampliacions que siguin d'interès per tal d'investigar la millor manera d'interaccionar amb robots col·laboratius.

7. Conclusions finals

Aquest apartat recull les conclusions extretes al finalitzar el treball de fi de grau, així com l'assoliment de les competències tècniques, el treball futur i la valoració personal.

7.1. Conclusions tècniques

Amb la realització d'aquest treball de fi de grau s'han aconseguit implementar dues interfícies gràfiques d'usuari que compleixen els objectius que es van plantejar a l'inici del projecte.

Per una banda, s'ha aconseguit integrar una URCap (un node de programació) implementada des de zero a la interfície de programació Polyscope. Permet programar i gestionar el *workflow* d'una producció industrial d'una manera intuïtiva, mitjançant blocs parametritzables que constitueixen el *workflow* en qüestió.

Durant la fase de proves de la URCap es va observar que Polyscope està més enfocat a interfícies gràfiques estàtiques (més aviat senzilles) i no tan dinàmiques (i complexes). Doncs, a Polyscope li costa gestionar vistes amb "molts" components visuals (en aquest cas JLabels que representen els blocs que constitueixen el *workflow*). Això no es va poder saber fins realitzar aquestes proves, perquè, durant tot el desenvolupament es va utilitzar el simulador de Polyscope que s'executava a l'ordinador de l'estudiant i no donava cap tipus de problema. Per tant, com a conclusió: no es pot esperar que el *hardware* d'un robot industrial sigui tan potent com el d'un ordinador.

Tot i això, el resultat de la URCap és molt bo i compleix les expectatives que es tenien en un inici, però queda treball pendent. Per exemple, intentar disminuir el nombre d'elements visuals de la interfície gràfica, fusionant alguns dels blocs del *workflow*.

D'altra banda, s'ha aconseguit desenvolupar una aplicació Android des de zero que es connecta via *socket* amb el robot. Permet: rebre informació sobre l'estat del robot, rebre informació sobre l'estat del programa, controlar l'execució del programa, poder visualitzar i filtrar les variables globals del programa (KPIs, de l'anlgès *Key Performance Indicator*), poder enviar missatges al robot, notificar els *popups* que s'executen al robot i notificar parades d'emergència o de protecció.

Cal destacar que un dels punts forts de l'aplicació és que permet un canal de comunicació entre aplicació – robot de dos sentits. Des de l'aplicació es poden enviar missatges de text al robot i des del robot es poden enviar missatges de text a l'aplicació (quan el robot executa un *popup* aquest es notifica al dispositiu mòbil de l'usuari, mostrant el tipus del *popup* i el seu missatge de text).

Les dues interfícies gràfiques s'han desenvolupat seguint la Norma ISA-101, intentant minimitzar la càrrega visual i cognitiva que l'usuari experimenta mentre interactua amb dites interfícies gràfiques.

Finalment, s'ha aconseguit definir una primera versió d'ontologia que dóna semàntica al conjunt d'operacions i macro-moviments que gestionen els productes desenvolupats. No s'ha disposat del temps suficient per poder desenvolupar una versió final de l'ontologia, tot i això s'ha deixat una bona base per on començar.

7.2. Competències tècniques

- **CCO1.1**: Avaluar la complexitat computacional d'un problema, conèixer estratègies algorísmiques que puguin dur a la seva resolució, i recomanar, desenvolupar i implementar la que garanteixi el millor rendiment d'acord amb els requisits establerts. [Bastant]

    - Durant les fases de desenvolupament dels productes sempre s'ha prioritzat l'eficiència del codi implementat, com s'ha especificat en els apartats corresponents.

- **CCO1.3**: Definir, avaluar i seleccionar plataformes de desenvolupament i producció hardware i software per al desenvolupament d'aplicacions i serveis informàtics de diversa complexitat. [Una mica]

    - Una part molt important del projecte va ser decidir quines plataformes o quins programaris utilitzar per desenvolupar les interfícies gràfiques d'usuari. Doncs, vam escollir les que vam creure més adequades per la seva finalitat principal. La URCap instal·lada a la consola de programació del robot permet programar i gestionar el *workflow* del procés industrial d'una manera molt intuïtiva. D'altra banda, l'aplicació Android permet monitoritzar tot el procés mentre s'està executant, sense la necessitat de que l'operador estigui "lligat" a la zona de producció industrial.

- **CCO2.1**: Demostrar coneixement dels fonaments, dels paradigmes i de les tècniques pròpies dels sistemes intel·ligents, i analitzar, dissenyar i construir sistemes, serveis i aplicacions informàtiques que utilitzin aquestes tècniques en qualsevol àmbit d'aplicació. [Bastant]

    - S'ha analitzat, dissenyat i construït dues aplicacions informàtiques amb una primera base semàntica amb la finalitat de que en futurs projectes es pugui afegir interacció per ordres vocals.

- **CCO2.2**: Capacitat per a adquirir, obtenir, formalitzar i representar el coneixement humà d'una forma computable per a la resolució de problemes mitjançant un sistema informàtic en qualsevol àmbit d'aplicació, particularment en els que estan relacionats amb aspectes de computació, percepció i actuació en ambients o entorns intel·ligents. [En profunditat]

    - La definició de l'ontologia cobreix aquesta competència. Tot i que en un principi teníem pensat poder-hi invertir més temps, l'ontologia presentada dota als productes d'una primera semàntica, que en un futur es pot escalar i definir amb més detall.

- **CCO2.3**: Desenvolupar i avaluar sistemes interactius i de presentació d'informació complexa, i la seva aplicació a la resolució de problemes de disseny d'interacció persona computador. [En profunditat]

    - Aquesta competència la cobreix el disseny de les dues interfícies gràfiques. Doncs, s'han desenvolupat i avaluat dos sistemes interactius que presenten informació complexa d'una manera intuïtiva per a que la interacció persona – robot es vegi afavorida.

7.3. Treball futur

Aquest treball de fi de grau obra moltes oportunitats de millorar-lo i escalar-lo en un futur. Doncs, des d'un bon inici aquest projecte estava pensat per ser la base de posteriors treballs de fi de grau o de màster amb la finalitat de que aquests poguessin afegir interacció amb ordres vocals.

A banda d'això, els productes desenvolupats poden escalar molt més, oferint altres funcionalitats o millorant les que s'han implementat en aquest treball de fi de grau.

L'ontologia presentada és una primera versió, per tant, s'hauria de definir amb més detall en futurs projectes si es vol aconseguir la interacció amb ordres vocals.

Doncs, amb la base que presenta aquest treball de fi de grau s'obre un ampli ventall de possibilitats per millorar la interacció entre humans i robots industrials, l'objectiu principal del projecte.

7.4. Valoració personal

Personalment aquest treball m'ha aportat molts coneixements sobre el món de la robòtica, un món que sempre m'ha apassionat. També, m'ha donat l'oportunitat de treballar amb un robot col·laboratiu real de Universal Robots.

Me n'he adonat de l'important paper que juga la interacció entre humans i robots en una producció col·laborativa i dels molts beneficis que implica tenir un bon canal de comunicació i d'operació.

Cal destacar que en un inici no sabia absolutament res sobre Universal Robots ni desenvolupament d'aplicacions Android, doncs, durant la realització d'aquest treball he adquirit molts coneixements nous que estic segur que en un futur em seran d'utilitat.

Estic content amb resultat final del treball, perquè ha complert els objectius i requeriments que es van presentar en un inici i deixa una base bastant sòlida per futurs projectes.

Referències

[1] Universal Robots, «Universal Robots,» [En línia]. Available: https://www.universal-robots.com/. [Últim accés: 26 Febrer 2021].

[2] IDEAI-UPC, «IDEAI-UPC. Intelligent Data Science and Artificial Intelligence Research Center,» [En línia]. Available: https://ideai.upc.edu/en. [Últim accés: 26 Febrer 2021].

[3] Recerca, Desenvolupament i Innovació. RDI, «Looming Factory,» [En línia]. Available: https://rdi.upc.edu/ca/ssri/projectes-institucionals/sectors-emergents-projectes/looming-factory. [Últim accés: 26 Febrer 2021].

[4] R. O. Eriksen, «Implementation and evaluation of movement primitives and a graphical user interface for a collaborative robot,» [En línia]. Available: https://upcommons.upc.edu/handle/2117/340118. [Últim accés: 26 Febrer 2021].

[5] A. Chacon, P. Ponsa i C. Angulo, «On Cognitive Assistant Robots for Reducing Variability in Industrial Human-Robot Activities,» 26 Juliol 2020. [En línia]. Available: https://www.researchgate.net/publication/343235556_On_Cognitive_Assistant_Robots_for_Reducing_Variability_in_Industrial_Human-Robot_Activities. [Últim accés: 26 Febrer 2021].

[6] S. P. Cobas, «Strategies for remote control and teleoperation of a UR robot,» 19 Juny 2020. [En línia]. Available: https://upcommons.upc.edu/handle/2117/333559. [Últim accés: 26 Febrer 2021].

[7] ESAII, «Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial,» [En línia]. Available: https://esaii.upc.edu/ca. [Últim accés: 26 Febrer 2021].

[8] Universal Robots Academy, «CB3 e-Learning,» [En línia]. Available: https://academy.universal-robots.com/free-e-learning/cb3-e-learning/. [Últim accés: 26 Febrer 2021].

[9] Universal Robots+, «URCap API Overview,» [En línia]. Available: https://plus.universal-robots.com/urcap-basics/api-overview/. [Últim accés: 20 03 2021].

[10] Wikipedia, «Android (operating system),» [En línia]. Available: https://en.wikipedia.org/wiki/Android_(operating_system). [Últim accés: 15 05 2021].

[11] Android Studio, «Guía del usuario,» [En línia]. Available: https://developer.android.com/studio/intro?hl=es-419. [Últim accés: 15 05 2021].

[12] Android Studio, «Android Studio Guide,» [En línia]. Available: https://developer.android.com/guide. [Últim accés: 15 05 2021].

[13] Universal Robots Support, «Script Manual - CB-Series - SW3.14,» [En línia]. Available: https://www.universal-robots.com/download/manuals-cb-series/script/script-manual-cb-series-sw314/. [Últim accés: 26 Febrer 2021].

[14] ROS, «ROS,» [En línia]. Available: https://www.ros.org/. [Últim accés: 26 Febrer 2021].

[15] CS-GEI-FIB, «Intel·ligència Artificial (GRAU-IA),» [En línia]. Available: https://www.cs.upc.edu/~bejar/ia/transpas/teoria/3-RC3-Ontologias.pdf. [Últim accés: 28 Febrer 2021].

[16] Protégé, «Protégé,» [En línia]. Available: https://protege.stanford.edu/. [Últim accés: 28 Febrer 2021].

[17] KEMLG, «Grup d'Enginyeria del Coneixement i Aprenentatge Automàtic. KEMLG,» [En línia]. Available: https://kemlg.upc.edu/ca. [Últim accés: 27 Febrer 2021].

[18] GREC, «Grup de Recerca en Enginyeria del Coneixement,» [En línia]. Available: https://www.fib.upc.edu/ca/recerca/grups-de-recerca/grec-grup-de-recerca-en-enginyeria-del-coneixement. [Últim accés: 27 Febrer 2021].

[19] SIC, «SIC - Sistemes Intel·ligents de Control,» [En línia]. Available: https://futur.upc.edu/SIC. [Últim accés: 27 Febrer 2021].

[20] CS2AC-UPC, «Centre de Recerca en Supervisió, Seguretat i Control Automàtic,» [En línia]. Available: https://cs2ac.upc.edu/ca. [Últim accés: 27 Febrer 2021].

[21] ISA - International Society of Automation, «ISA101, Human-Machine Interfaces,» [En línia]. Available: https://www.isa.org/standards-and-publications/isa-standards/isa-standards-committees/isa101. [Últim accés: 28 Febrer 2021].

[22] Wikipedia, «Metodologia àgil,» [En línia]. Available: https://ca.wikipedia.org/wiki/Metodologia_%C3%A0gil. [Últim accés: 01 Març 2021].

[23] Lucidchart, «Scrum for one: A tutorial on adapting Agile Scrum methodology for individuals,» [En línia]. Available: https://www.lucidchart.com/blog/scrum-for-one. [Últim accés: 19 6 2021].

[24] Trello, «Trello,» [En línia]. Available: https://trello.com/. [Últim accés: 01 Març 2021].

[25] GanttProject, «GanttProject,» [En línia]. Available: https://www.ganttproject.biz/. [Últim accés: 5 Març 2021].

[26] Hays, «Hays,» [En línia]. Available: https://guiasalarial.hays.es/trabajador/home. [Últim accés: 12 Març 2021].

[27] MediaMarkt, «MediaMarkt,» [En línia]. Available: https://www.mediamarkt.es/es/product/_disco-duro-ssd-interno-de-480-gb-sandisk-ssd-plus-hasta-535-mb-s-1336397.html. [Últim accés: 12 Març 2021].

[28] Universal Robots, «Universal Robots Support Download,» [En línia]. Available: https://www.universal-robots.com/download/manuals-cb-series/user/ur3/314/user-manual-ur3-cb-series-sw314-spanish-es/. [Últim accés: 13 Març 2021].

[29] Tarifa luz hora, «Tarifa luz hora,» [En línia]. Available: https://tarifaluzhora.es/?tarifa=normal&fecha=12%2F03%2F2021. [Últim accés: 12 Març 2021].

[30] The Office Barcelona, «The Office Barcelona,» [En línia]. Available: https://www.theofficebcn.com/. [Últim accés: 14 Març 2021].

[31] Scratch, «Scratch,» [En línia]. Available: https://scratch.mit.edu/about. [Últim accés: 1 6 2021].

[32] Universal Robots, «Forum Universal Robots,» [En línia]. Available: https://forum.universal-robots.com/t/java-version-for-ursim-3-6/2435. [Últim accés: 1 06 2021].

[33] Universal Robots, «Principle of URCaps integration in PolyScope,» [En línia]. Available: https://plus.universal-robots.com/urcap-basics/principle-of-urcaps-in-polyscope/. [Últim accés: 2 6 2021].

[34] Universal Robots Support, «Overview of client interfaces,» [En línia]. Available: https://www.universal-robots.com/articles/ur/interface-communication/overview-of-client-interfaces/. [Últim accés: 4 6 2021].

[35] Universal Robots Support, «Remote Control Via TCP/IP,» [En línia]. Available: https://www.universal-robots.com/articles/ur/interface-communication/remote-control-via-tcpip/. [Últim accés: 7 6 2021].

[36] Android, «Introduction to Activities,» [En línia]. Available: https://developer.android.com/guide/components/activities/intro-activities. [Últim accés: 10 6 2021].

[37] Android, «Services overview,» [En línia]. Available: https://developer.android.com/guide/components/services. [Últim accés: 10 6 2021].

[38] Android, «Bound services overview,» [En línia]. Available: https://developer.android.com/guide/components/bound-services. [Últim accés: 10 6 2021].

[39] Android, «Background Execution Limits,» [En línia]. Available: https://developer.android.com/about/versions/oreo/background. [Últim accés: 10 6 2021].

Annexes

Integració de coneixements

Aquest treball de fi de grau integra coneixements de diferents àrees que s'han estudiat i treballat durant el grau universitari.

- Els coneixements adquirits a l'assignatura de PROP per a la correcta gestió del projecte.
- Els coneixements adquirits a l'assignatura de IES per estructurar degudament en classes tot el codi, així com definir la representació UML. També per dissenyar la implementació de cares a futures millores o complementacions.
- L'eficiència i la gestió i elecció de les estructures de dades adients són de rellevant importància, perquè el *hardware* del robot no té la potència de càlcul, memòria, etc. D'un ordinador, doncs, el codi ha de ser eficient per no alentir l'execució, el que implicaria no assolir l'objectiu d'una interacció robot-humà fluida. Coneixements adquirits a assignatures com EDA, PRO1, PRO2, etc.
- Els coneixements adquirits a l'assignatura de IA per a la definició de l'ontologia.
- Els coneixements adquirits a l'assignatura de IDI pel disseny d'interfícies gràfiques d'usuari.

D'altra banda, s'han adquirit nous coneixements:

- Entrar al món de Universal Robots, totalment desconegut per a l'estudiant en un inici.
    - Aprendre com funciona el robot, com programar-lo, quines parts el componen, etc.
    - Aprendre la base del funcionament del *software* del robot, Polyscope.
    - Aprendre a implementar una URCap utilitzant el SDK que ofereix Universal Robots.
    - Aprendre URScript i com Polyscope transforma el programa que conté al seu arbre de programa en URScript per executar-lo.
    - Aprendre com comunicar-se amb el robot, quines opcions hi ha, com es rep la informació, etc.
- Entrar al món d'Android, totalment desconegut per a l'estudiant en un inici:
    - Aprendre les bases del desenvolupament Android mitjançant la IDE Android Studio.
    - Com gestionar les diferents activitats o vistes.
    - Com passar informació entre activitats.
    - Com modificar dinàmicament la informació que mostra cada un dels components que hi ha a les diferents activitats.

En definitiva, aquest treball de fi de grau integra coneixements de diferents disciplines que tenen com a finalitat assolir el principal objectiu del projecte: millorar la interacció entre humans i robots industrials col·laboratius.

Identificació de lleis i regulacions

Degut a que el projecte no treballa ni registra cap tipus de dades personals (informacions de contacte, gravacions, missatges, àudios, etc.) no està sotmès a cap tipus de llei de protecció de dades. Totes les tecnologies utilitzades són programari lliure de codi obert.

Les interfícies gràfiques d'usuari s'han dissenyat seguint la Norma ISA-101, que pretén marcar una sèrie de convencions i normes a l'hora del disseny d'interfícies HMI (de l'anglès *Human-Machine Interfaces*). Aquesta normativa és de pagament, per tant, ens hem basat en la informació que estava disponible per internet i la que ens ha subministrat Pere Ponsa Asensio.

Enllaços GitHub

Aquest annex conté els enllaços als repositoris on es troba el codi font dels productes desenvolupats:

- URCap: https://github.com/Wenry19/TFG_PolyscopeNodes
- App Android: https://github.com/Wenry19/TFG_AndroidAppUR
- Ontologia: https://github.com/Wenry19/TFG_Ontology

Javadocs

A l'apartat de material addicional s'ha entregat un fitxer zip que conté els javadocs que documenten el codi font dels productes desenvolupats.

Especificació dels paquets descodificats

Aquest annex especifica el format amb el qual s'envien els paquets d'informació de la Interfície Primària dels robots de Universal Robots (versió 3.14). Només s'especifiquen els que s'han utilitzat en aquest treball de fi de grau. Tota la informació d'aquest annex s'ha extret dels documents adjunts de [35].

Robot mode data (Sub Package of Robot State Message)	
int packageSize	Length of Sub Package: total length of subpackage including this field
unsigned char packageType = ROBOT_STATE_PACKAGE_TYPE_ROBOT_MODE_DATA = 0	
uint64_t timestamp	
bool isRealRobotConnected	
bool isRealRobotEnabled	
bool isRobotPowerOn	
bool isEmergencyStopped	
bool isProtectiveStopped	
bool isProgramRunning	
bool isProgramPaused	
unsigned char robotMode	RobotModes
unsigned char controlMode	ControlModes
double targetSpeedFraction	
double speedScaling	
double targetSpeedFractionLimit	
unsigned char reserved	

Joint data (Sub Package of Robot State Message)	
int packageSize	
unsigned char packageType = ROBOT_STATE_PACKAGE_TYPE_JOINT_DATA = 1	
for each joint:	
double q_actual	
double q_target	
double qd_actual	
float I_actual	
float V_actual	
float T_motor	
float T_micro	**Deprecated - ignore**
uint8_t jointMode	JointModes
end	

Tool data (Sub Package of Robot State Message)	
int packageSize	Length of Sub Package: total length of subpackage including this field
unsigned char packageType = ROBOT_STATE_PACKAGE_TYPE_TOOL_DATA = 2	
unsigned char analogInputRange0	
unsigned char analogInputRange1	
double analogInput0	
double analogInput1	
float toolVoltage48V	
unsigned char toolOutputVoltage	
float toolCurrent	
float toolTemperature	
uint8_t toolMode	ToolModes

Masterboard data (Sub Package of Robot State Message)

int packageSize	Length of Sub Package: total length of subpackage including this field
unsigned char packageType = ROBOT_STATE_PACKAGE_TYPE_MASTERBOARD_DATA = 3	
int digitalInputBits	
int digitalOutputBits	
unsigned char analogInputRange0	
unsigned char analogInputRange1	
double analogInput0	
double analogInput1	
char analogOutputDomain0	
char analogOutputDomain1	
double analogOutput0	
double analogOutput1	
float masterBoardTemperature	
float robotVoltage48V	
float robotCurrent	
float masterIOCurrent	
unsigned char safetyMode	See SafetyModes sheet
uint8_t InReducedMode	
char euromap67InterfaceInstalled	
if euromap67 interface is installed, also the following:	
uint32_t euromapInputBits	
uint32_t euromapOutputBits	
float euromapVoltage24V	
float euromapCurrent	
end	
uint32_t (Used by Universal Robots software only)	
uint8_t operationalModeSelectorInput	
uint8_t threePositionEnablingDeviceInput	
unsigned char (Used by Universal Robots software only)	

Robot Message - PopupMessage (Primary client only)

int messageSize	
unsigned char messageType = MESSAGE_TYPE_ROBOT_MESSAGE = 20	
uint64_t timestamp	
char source	MessageSources
char robotMessageType = ROBOT_MESSAGE_TYPE_POPUP = 2	
unsigned int requestId	
unsigned int requestedType	RequestedTypes
bool warning	
bool error	
bool blocking	
uint8_t popupMessageTitleSize	
charArray popupMessageTitle	
charArray popupTextMessage	

GlobalVariablesSetupMessage (Primary client only)

int messageSize	
unsigned char messageType = MESSAGE_TYPE_PROGRAM_STATE_MESSAGE = 25	
uint64_t timestamp	
char robotMessageType = PROGRAM_STATE_MESSAGE_TYPE_GLOBAL_VARIABLES_SETUP = 0	
uint16_t startIndex	
charArray variableNames	List of names separated by new line character ('\n' character). Examp

GlobalVariablesUpdateMessage (Primary client only)

int messageSize	
unsigned char messageType = MESSAGE_TYPE_PROGRAM_STATE_MESSAGE = 25	
uint64_t timestamp	
char robotMessageType = PROGRAM_STATE_MESSAGE_TYPE_GLOBAL_VARIABLES_UPDATE = 1	
uint16_t startIndex	
unsigned char variableValues	ValueTypes

Each variable value contains type byte, data, and terminating new line character (\n character). Example - two variables of ty