UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Robot-vehicle adaptatiu per a l'Olympic Robotic Challenge

Treball Fi de Màster

Màster Universitari en Enginyeria Informàtica

Autor: Efren Camarasa Carreres

Tutor: Joan Fons i Cors

Curs 2017/2018

Resum

La complexitat del programari i la connexió amb altres de dinàmics i canviants fa necessari el disseny de programari flexible i adaptable per a poder funcionar correctament i sense interrupció alguna davant de situacions de no disponibilitat d'entorns connectats i canviants.

El projecte presentat en aquest document té com a finalitat el disseny de l'adaptabilitat d'un vehicle mitjançant una de les tècniques disponibles en l'actualitat com són els bucles MAPE-K. Per a exposar aquest disseny a una situació real i pràctica, l'aplicarem a un vehicle que participa a una competició existent com és l'ORC (*Olympic Robotic Challenge*) celebrada en la Universitat Politècnica de València.

A través d'aquest document s'exposa com ha sigut dissenyada la part d'enginyeria conceptual que té com a objectiu l'especificació dels components que formen la solució i com aquests són posats en funcionament sobre un prototip. La part d'implementació física no és desenvolupada en aquest projecte, aquest objectiu quedaria per a una ampliació posterior.

Paraules clau: disseny, adaptabilitat, vehicle, MAPE-K, ORC

Resumen

La complejidad del *software* y la conexión con otros dinámicos y cambiantes hace necesario el diseño de *software* flexible y adaptable para poder funcionar correctamente sin interrupción alguna delante de situaciones de no disponibilidad de entornos conectados y cambiantes.

El proyecto presentado en este documento tiene como finalidad el diseño de la adaptabilidad de un vehículo mediante una de la técnicas disponibles en la actualidad como son los bucles MAPE-K. Para exponer este diseño a una situación real y práctica, se aplicará a un vehículo que participa en una competición existente como es la ORC (*Olympic Robotic Challenge*) celebrada en la Universitat Politècnica de València.

A través de este documento se expone cómo ha sido diseñada la parte de ingeniería conceptual que tiene como objetivo la especificación de los componentes que forman la solución y como estos son puestos en funcionamiento sobre un prototipo. La parte de implementación física no es desarrollada en este proyecto, este objetivo quedaría para una ampliación posterior.

Palabras clave: diseño, adaptabilidad, vehículo, MAPE-K, ORC

Abstract

The complexity of the software and the connection with other dynamic and changing technologies makes it necessary to design flexible and adaptable software capable of functioning correctly without any interruption in situations in which connected and changing environments are not available

The project presented in this document has the purpose of designing the adaptability of a vehicle using one of the techniques currently available, such as the MAPE-K loops. To expose this design to a real and practical situation, it will be applied to a vehicle which participates in an existing competition such as the ORC (Olympic Robotic Challenge) celebrated at the Universitat Politècnica de València.

Through this document we explain how the conceptual engineering part has been designed. It aims to specify the components that form the solution and how they are put into operation on a prototype. The physical implementation part is not developed in this project, this objective would be for a later work.

Key words: design, adaptability, vehicle, MAPE-K, ORC

Índex general

Índex de figures

Índex de taules

Índex de codis

Capítol 1

Introducció

En aquest capítol es mostra la motivació per a dur a terme el tipus de solució emprada i quins són els objectius que es volen complir al finalitzar el projecte.

1.1 Motivació

Els sistemes tecnològics actuals estan composats per un gran nombre de components o depenen de serveis externs els quals no sempre estan disponibles o no són accessibles a causa de la seva dinamicitat.

Degut a aquests escenaris, és necessari que el programari desenvolupat propose un nou enfoc. El progamari que es desenvolupa actualment no està preparat per a tindre present totes les situacions i ser capaç d'adaptar-se a circumstàncies anòmales o intentar previndre errors els quals poden provocar conseqüències devastadores. Un exemple clar són els vehicles autònoms, els quals deuen ser capaços de previndre circumstàncies desfavorables o recuperar-se davant de fallades provocades pels sistemes que els componen.

Per tant, el nou programari que es desenvolupe deu ser capaç d'obtenir els paràmetres necessaris d'un altre component o servei alternatiu per a que el sistema tecnològic no pare mai d'oferir la seva funcionalitat.

1.2 Problemàtica

Pels motius exposat en la secció 1.1 és necessari emprar noves estratègies i eines que proporcionen sistemes coneguts com a sistemes amb capacitats de computació autònoma.

Per a poder dur a terme el desenvolupament d'aquest tipus de programari és necessari canviar l'enfoc o el paradigma de la programació. Habitualment el programari desenvolupat es realitzat per un paradigma anomenat funcional, el qual determina les accions que ha de dur a terme el programari. En canvi, una programació dirigida a la autoadaptació es centra en especificar com el programari deu canviar per a satisfer els nous requisits.

1.3 Objectius

L'objectiu d'aquest projecte és dissenyar un programari adapatatiu i instanciar-lo en un dels vehicles dissenyats per a la competició ORC mitjançant una de les teories de control disponibles que fan possible aquest tipus de programari, els bucles MAPE-K.

Aquesta solució deu ser capaç de reaccionar front a esdeveniments que provoquen fallades durant les dues proves que realitza el vehicle, la resolució d'un laberint i la carrera de velocitat. Per a poder dur a terme aquesta solució el sistema s'ha d'adonar quan ocorren aquests esdeveniments i com reaccionar davant d'ells per a que el vehicle no entre en una situació de bloqueig durant les proves.

1.4 Metodologia

Per a la realització d'aquest treball s'ha seguit la metodologia esperada amb un projecte relacionat amb el programari. Primerament es va especificar quin serien els components que compondrien el vehicle per a les proves que anaven a ser dutes a terme. Una vegada especificats els components necessaris, un diagrama de components va ser realitzat per a relacionar aquests components entre ells i així especificar també les estratègies que estarien disponibles per a la resolució de les proves.

Amb aquest esquema plasmat, era l'hora d'utilitzar l'eina disponible i començar a especificar-lo de la manera que aquesta ens permet. Una vegada acceptat l'esquema per l'eina es va generar el codi necessari per a poder dur a terme la programació dels components del vehicle de manera individual, així com les regles d'adaptació i els escenaris d'adaptació davant de situacions adverses.

1.5 Estructura del document

Aquest document està organitzat en els següents 9 capítols:

1. **Introducció:** Primer capítol del document on s'exposa la motivació per la qual desenvolupar un programari orientat a la adaptabilitat, quina és la situació actual, com afecta a aquest tipus de programari i quin es l'objectiu que es vol alcançar amb aquest projecte.

2. **Context tecnològic:** Capítol on es mostra la tecnologia emprada per a la implementació de la solució. Es descriu tant l'entorn de programació utilitzat com els *frameworks* que permeten una solució adaptativa. Per últim una explicació breu de les solucions que existeixen actualment per a la realització d'aquest tipus de programari.

3. **Cas d'estudi:** Breu resum on s'explica la popularització del moviment *maker* i com ha influït en les competicions de robòtica. A més de l'explicació de les proves que deu dur a terme el vehicle dissenyat, així com els esquemes de disseny del vehicle i els diagrames de flux.

4. **Computació autònoma:** Quart capítol del document que exposa en que consisteix la computació autònoma i quina estratègia/solució és utilitzada per a obtenir un programari adaptatiu.

5. **Anàlisis del problema:** Mostra els esquemes dissenyats al inici del projecte per a saber la relació que existeix entre els components que conformen el vehicle. També es mostra quines situacions adverses es tenen en compte per a que el vehicle puga adaptar-se front a elles.

6. **Disseny de la solució:** Llista les estratègies pensades per a la resolució de les proves i mostra els diagrames de tots els elements que componen el vehicle.

7. **Implementació:** Explicació dels quatre components desenvolupats que componen la solució adaptativa realitzada, així com fragments de codis que expliquen la lògica d'aquests.

8. **Proves de la solució:** Demostració d'un parell d'exemples on ocorren esdeveniments no desitjats i com influeixen sobre els escenaris. Prova de situacions mitjançant una eina de programari desenvolupada per a que no siga necessari tindre el vehicle físicament.

9. **Conclusions i treballs futurs:** Últim capítol on s'exposa si els objectius han sigut assolits i idees per a ampliacions futures. A més d'una valoració personal respecte a la realització i el resultat del projecte.

Capítol 2

Context tecnològic

En aquest capítol es revisen les tecnologies utilitzades per al desenvolupament de la solució adaptativa. Es revisen tant l'entorn de programació com el llenguatge utilitzat. A més es mostren els dos *frameworks* emprats que permeten l'especificació dels components de la solució i que aquests puguen ser extrets o incorporats dinàmicament. Per últim, s'exposen algunes propostes teòriques i *frameworks* alternatius als utilitzats.

2.1 Desenvolupament: llenguatge, frameworks i entorn

Per a la realització de la solució adapativa és necessari la utilització d'un llenguatge de programació con és *Java* que ens permeta definir la lògica. A més d'un llenguatge de programació, la utilització dels *frameworks* que ens permeten definir el sistema i les seves característiques dona lloc a la utilització d'un entorn de programació per poder utilitzar-ho tot conjuntament.

2.1.1 Java

Llenguatge de programació utilitzat per al desenvolupament de la solució, ja que el codi generat per el *framework* PROTeus ens ofereix el codi autogenerat en aquest llenguatge.

2.1.2 PROTeus

Framework utilitzat per a l'especificació del diagrama de components adaptatius i la creació dels fitxers on s'incorpora la programació de cadascun dels components. Ens permet la creació de components adaptatius i la relació entre aquests en diferents escenaris. Proporciona editors per a la creació d'aquests components (figura 2.1), així com a quins esdeveniments (RT) atenen les regles d'adaptació (figura 2.2).

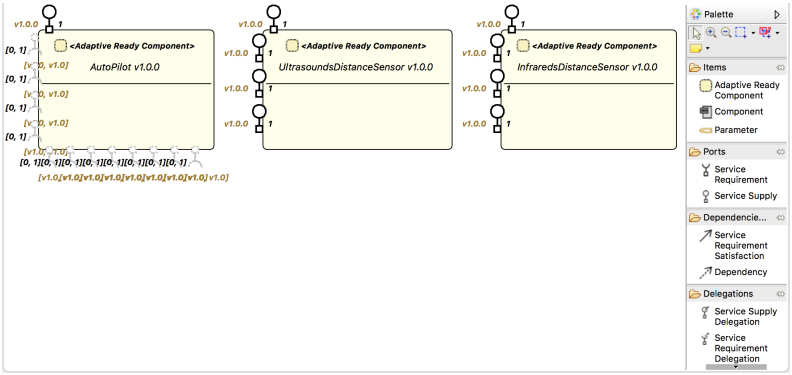

Figura 2.1: Editor de components

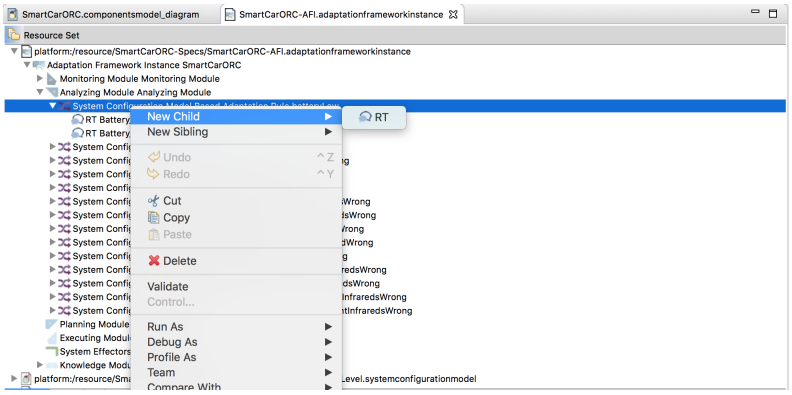

Figura 2.2: Editor de regles d'adaptació

Aquest *framework* és el resultat d'un projecte per part del *Centre PROS* del grup *TATAMI* del qual és membre Joan Fons i Cors, tutor d'aquest projecte. Aquest centre centra part de la seva recerca a temes de computació

autònoma i construïren fa no molt de temps aquesta eina o *framework* que dona suport a un procés de disseny i desenvolupament de sistemes autoadaptatius.

2.1.3 OSGi

Framework que permet que els components desenvolupats puguen ser extrets o incorporats dinàmicament per a permitir que el programari siga adaptatiu. Aquestes accions poden ser realitzades perquè els components de la solució (anomenats *bundles*), ofereixen un cicle de vida el qual permet que el component estiga actiu, parat o inclús no disponible [1].

2.1.4 Eclipse IDE

Per a desenvolupar la solució ha sigut necessària la utilització d'un IDE (*Integrated Development Environment*) que facilatara la incorporació de *plugins* per a utilitzar els *frameworks* necessaris. Eclipse ens permet crear llocs de treballs diferents (*workspaces*) que permeten la incorporació de diversos projectes relacionats entre sí. La figura 2.3 mostra aquest entorn amb els projectes creats per a albergar la solució del projecte.

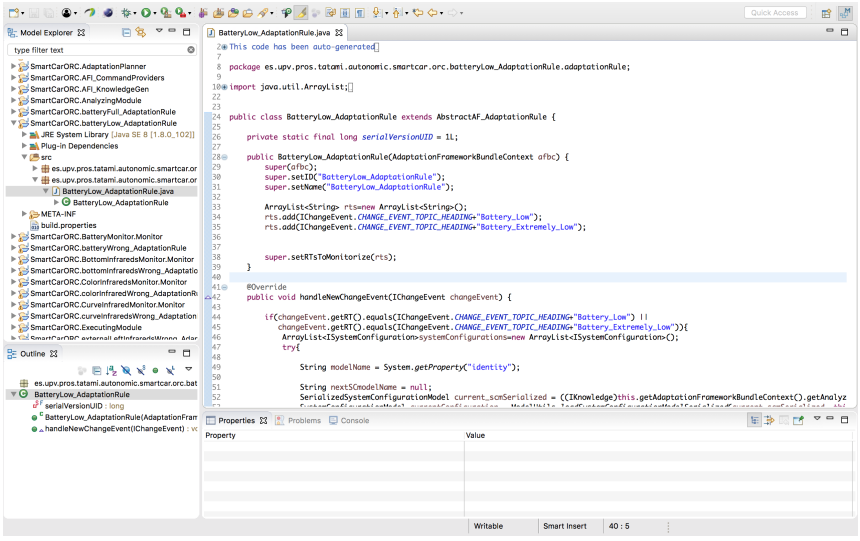

Figura 2.3: Entorn Eclipse IDE

2.2 Comunicacions

La solució duta a terme es divideix en diferents fases que realitzen tasques específiques. Per a que aquestes fases es puguen comunicar entre elles és necessari la utilització de protocols de comunicació. Per a aquest projecte s'ha optat per utilitzar el protocol de comunicacions *MQTT*.

2.2.1 MQTT

Protocol de missatgeria que implementa l'escenari publicador-subscriptor a través d'elements *brokers* que actuen com a intermediaris. S'utilitza per a la comunicació entre diversos elements que formen la solució desenvolupada. L'escenari publicador-subscriptor permet la retransmissió de missatges a aquells elements que es subscriuen a un tema, els elements *brokers* són els encarregats de fer correctament aquesta retransmissió.

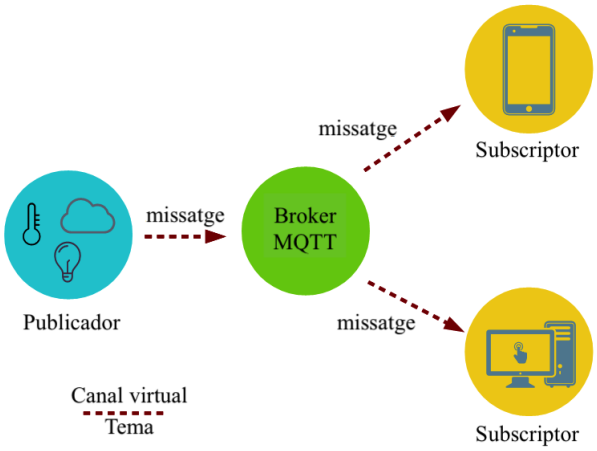

Figura 2.4: Escenari publicador-subscriptor

2.3 Computació autònoma: propostes teòriques i frameworks existents

La solució implementada no és l'única que existeix en el mercat actual. Hi ha diverses d'elles, encara que no moltes. Algunes d'elles són explicades a continuació.

2.3.1 FESAS

FESAS [2] és un *framework* orientat al desenvolupament de sistemes autoadaptatius. Es compon de dues parts. La primera d'elles és la denominada *FESAS Development Tool*, la qual ens permet definir la funcionalitat dels components que formen el sistema. La segona part és coneguda com a *FESAS Desing Tool*, ja que ens permet dissenyar el sistema i com els components interactuen entre sí.

Aquest *framework* està disponible per a la seva descàrrega en la següent direcció: https://fesas.bwl.uni-mannheim.de/download/. A més de la descàrrega, el lloc web mostra informació sobre el *framework* o la seva instal·lació.

2.3.2 Genie

Genies [3] és un *framework* per a aplicacions *web*. En una primera instància va ser creat per a un desenvolupament ràpid de *webs* relacionades amb bases de dades. Inclús així pot ser utilitzat per a qualsevol tipus d'aplicació *web*. Encara que siga un *framework* orientat a les aplicacions *web*, s'inclueix ja que ofereix sistemes autoadaptatius.

Aquest *framework* està disponible per a la seva descàrrega en la següent direcció: http://genie-framework.sourceforge.net/#d0e142. A més de la descàrrega, el lloc web mostra documentació sobre el *framework* i inclús les dependències amb altres llibreries per a la seva utilització.

2.3.3 Kevoree

Kevoree [4] és un *framework* que ofereix utilitats per al desenvolupament de sistemes distribuïts adaptables i reconfigurables en temps d'execució. Utilitza el seu propi *framework* de modelatge i diferents abstraccions per a disminuir la complexitat al desenvolupar la solució.

Aquest *framework* està disponible per a la seva descàrrega en la següent direcció: http://kevoree.org/download/download.html per a diverses plataformes. A més de la descàrrega, el lloc *web* proporciona una guia i un exemple per a crear un projecte.

Capítol 3

Cas d'estudi

En aquest capítol es mostra la situació inicial de la qual parteix el projecte i les restriccions que imposa l'ORC per a la construcció dels vehicles. A més s'expliquen en que consisteixen les proves per a les quals ha sigut dissenyat el sistema autoadaptatiu, els esquemes del vehicle i els diagrames de flux per a cadascuna de les proves.

3.1 Plantejament inicial

Amb la popularització del moviment *maker* (fes-ho tu mateix) cada vegada hi ha més competicions sobre robòtica i on participen un gran nombre de persones. A més, moltes d'elles estan orientades cap a un públic d'una edat jove per a així descobrir-los un nou món i dirigir-los cap a camps d'estudis relacionats amb les ciències, la tecnologia i sobretot la robòtica.

Per a participar en aquestes competicions, els participants realitzen el muntatge de xicotets robots/vehicles adequats per a la realització de les proves involucrades. Sol ser comú una guia de muntatge on es mostra un exemple d'un prototip d'un robot/vehicle participant. A més és freqüent que es proporcione un llistat amb els components per a la realització d'aquest prototip, on s'indica el component, un preu estimat i inclús un enllaç a una tenda *online*. Una vegada el prototip o ja el robot/vehicle final es realitzat, les persones o equips participants s'impliquen amb la programació per a la realització de les proves que componen la competició. Aquest programari

desenvolupat està enfocat a la realització de les proves i no es pensa mai amb les situacions anòmales que poden ocórrer, com per exemple la pèrdua de sensors per xocs o l'esgotament de la bateria.

En la UPV (Universitat Politècnica de València) cada any s'organitza una competició anomenada ORC composada per diverses proves. Abans de que la competició es realitze, tallers de formació són duts a terme per a que els participants tinguen una pressa de contacte amb el món de la robòtica si no ho havien fet abans. En aquests tallers es segueix el *kit* que la competició recomana [5], encara que no és obligatòria la utilització d'aquests components per a participar. Amb aquests components ensenyen als participants les idees bàsiques per a la construcció del vehicle seguint uns esquemes [6] per a la connexió dels components entre sí.

En els 2 anys que porta realitzant-se l'ORC en la UPV s'ha manifestat com el programari desenvolupat està enfocat a la resolució de les proves i no es té en compte les situacions no desitjades tal i com es deia abans. La pèrdua de components per col·lisions entre vehicles és molt freqüent i el bloqueig del vehicle per situacions no tingudes en compte és una altra causa de que els vehicles no responguen com el desenvolupadors esperaven.

Degut a aquestes circumstàncies es va pensar en dissenyar un programari adaptatiu en tal de que el vehicle no es quedara mai bloquejat i poguera acabar les proves dutes a terme i inclús crear estratègies que ens permeten realitzar les proves de la manera més ràpida possible, com per exemple reconeixent el circuit en la primera oportunitat de la prova i així poder fer-la més ràpidament en la segona oportunitat.

Per a que aquesta adaptabilitat siga possible hi ha tindre clar quin nivell d'adaptació es vol tindre en compte. En aquest cas es vol tindre un nivell d'adaptabilitat total, per tant hi ha que tenir en compte qualsevol situació no desitjada o anòmala possible, tant aquelles relacionades amb els sensors (mal funcionament o no disponibilitat), amb els actuadors, el nivell de bateria o el temps proporcionat per a la realització de les proves.

3.2 Requeriments

- **Maquinari:** Els requeriments de maquinari per a aquest projecte són aquells que permeten la construcció del vehicle per a la participació de la competició. Les dimensions i el pes del vehicle estan definits en el document on s'especifica la normativa de la competició [7]. En quant als sensors que composen el vehicle, les normes no especifiquen la utilització dels components exactes, per tant està permès qualsevol element sempre que no es superen les dimensions i el pes, a més d'elements que no comprometen la seguretat.

- **Programari:** Els requeriments del programari per a quest projecte són aquells que permeten la creació d'un programari adaptatiu que permeta la incorporació i l'extracció de components del programari per a que aquest permeta l'adaptabilitat del vehicle.

3.3 Proves implicades

3.3.1 Resolució del laberint

Prova de la competició ORC que té com a finalitat la resolució d'un laberint (figura 3.1) en el menor temps possible. Per a dur a terme aquesta prova és necessari la incorporació d'un sensor d'infraroig que ens permeta saber el color que hi ha marcat en les parets que componen el laberint (figura 3.2), ja que aquestes marques de colors ens permeten saber la direcció que deu agafar el vehicle per a trobar l'eixida i un altre sensor d'infraroig per a no col·lisionar amb la paret frontal. Com sol es depèn d'un sensor de distància i de sentit del gir, si aquest sensor falla o per alguna circumstància fora extret del vehicle, aquest ja no podria seguir amb prova.

Per tant, es podria incorporar un sensor d'ultrasons a la part davantera i dos sensors infraroig als laterals que ens permitiren mesurar la distància amb les parets i les obertures laterals per a realitzar la funcionalitat dels sensors anteriorment nomenats. Així, per exemple, si el vehicle funcionara amb aquests últims sensors podria saber quan està davant d'una paret, parar per a no col·lisionar amb ella, comprovar les obertures laterals i decidir cap a quin dels dos costats seguir o inclús anar cap enrere si es troba en un camí sense eixida sense tindre cap informació dels colors marcats en les parets.

Degut a que l'objectiu d'aquesta prova és resoldre el laberint en el menor temps possible, seria convenient la creació d'estratègies on la velocitat de la seva resolució puguera canviar segons l'estat del sensors. Per exemple, quan el temps de la prova està a punt de vèncer seria convenient una velocitat lo més alta possible. En canvi, si realitzant la prova, el nivell de bateria estiguera en nivells mínims seria convenient una velocitat mínima per a així estalviar bateria.

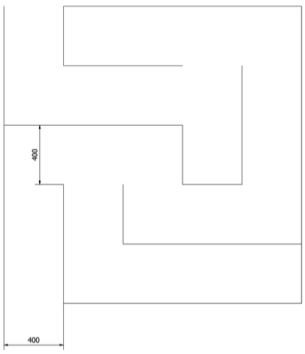

Figura 3.1: Trajectòria del laberint

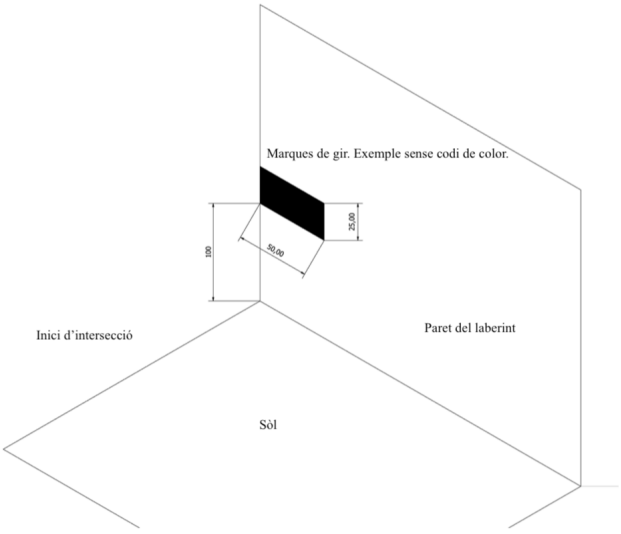

Figura 3.2: Marca de color que indica la direcció

3.3.2 Carrera de velocitat

Prova de la competició ORC que consisteix en recórrer un circuit marcat en el sòl en el menor temps possible. Aquest circuit està compost per una línia central de color negre i dos laterals de color roig que marquen els límits que no pot superar el vehicle (figura 3.3). A més també s'incorporen unes línies de colors roig, verd o blau entre la línia central i la lateral dretana que marquen l'inici i el final d'una corba i la seva obertura segons el seu color (figura 3.4).

Per a poder realitzar aquesta prova és almenys necessari un sensor d'infraroig al centre del vehicle el qual ens indique que estem dins del límits o sobre la línia central, a més d'un altre que ens indique que una corba s'aproxima. Si el sensor d'infraroig central té algun contratemps, el vehicle no sabria en cap moment on es troba, ja que es podria trobar fora dels límits i no adonar-se'n. Per tant, la incorporació de dos sensor d'infraroig laterals que monitoritzaren els límits seria una solució per a saber que el vehicle es troba dins del circuit i suplir la funcionalitat del sensor d'infraroig central.

Degut a que l'objectiu d'aquesta prova és recórrer el circuit en el menor temps possible, seria convenient la creació d'estratègies on la velocitat de la seva resolució puguera canviar segons l'estat del sensors. Per exemple, quan el temps de la prova està a punt de vèncer seria convenient una velocitat lo més alta possible. En canvi, si realitzant la prova, el nivell de bateria estiguera en nivells mínims seria convenient una velocitat mínima per a així estalviar bateria.

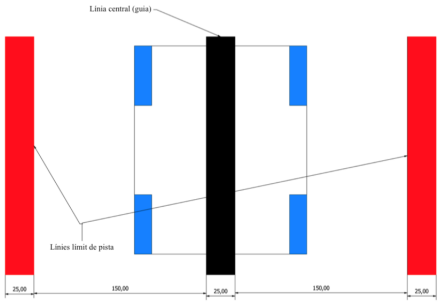

Figura 3.3: Línies del circuit de velocitat

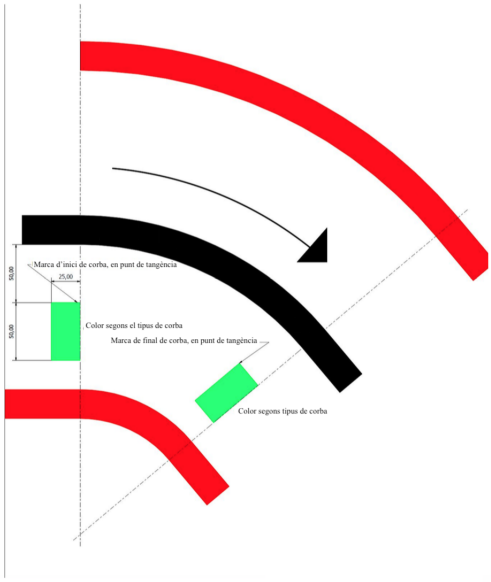

Figura 3.4: Indicadors d'inici i fi de corba

3.4 Esquemes del vehicle

Les següents figures mostren per separat els components per a la realització de les proves implicades.

En la figura 3.5 es mostra l'estructura del vehicle per a la prova de la resolució del laberint. Hi han sensors representats amb els mateix color a pesar de que son elements totalment diferents i això és degut a que aquests ofereixen la mateixa funcionalitat, per a que quan un d'ells falle, el sistema el substitueixca per un altre. Els sensors *LeftInfraredsDistanceSensor* i *RightInfraredsDistanceSensor* formen un sol sensor a efectes pràctics, ja que determinen cap a quina direcció deu girar el vehicle al igual que ofereix *InfraredsColorSensor*. Els sensor de color roig determinen la distància amb la paret frontal per a no col·lisionar, els sensors de color blau proporcionen informació sobre cap a quin costat deu girar el vehicle per trobar l'eixida i per últim, els elements de color verd són les rodes/motors que conformen el sistema de tracció.

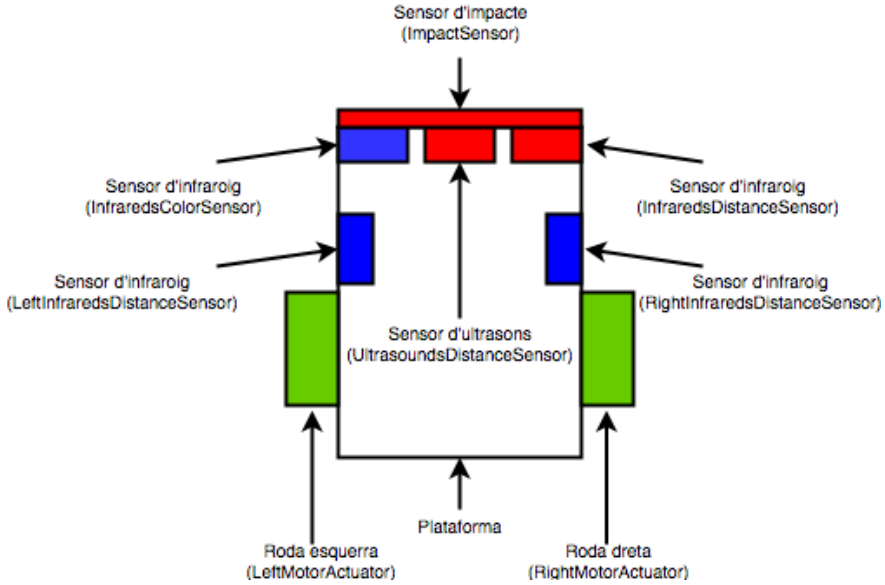

Figura 3.5: Estructura del vehicle per a la prova de la resolució del laberint

En la figura 3.6 es mostra l'estructura del vehicle per a la prova de la carrera de velocitat. Al igual que la figura anterior, hi ha sensors representats amb el mateix color pel mateix motiu. En aquest cas els sensors *LeftInfraredsLineSensor* i *RightInfraredsLineSensor* també actuen com a un sol sensor que ofereix la mateixa funcionalitat que el sensor *InfraredsLineSensor* que permet determinar si estem sobre la línia pintada en el sòl. Els sensors de color roig proporcionen informació sobre la situació del vehicle, és a dir, si està dins dels límits o no. El sensor de color blau informa de que una corba s'aproxima i els elements de color verds tenen la mateixa funcionalitat que la explicada en la figura anterior, la tracció del vehicle.

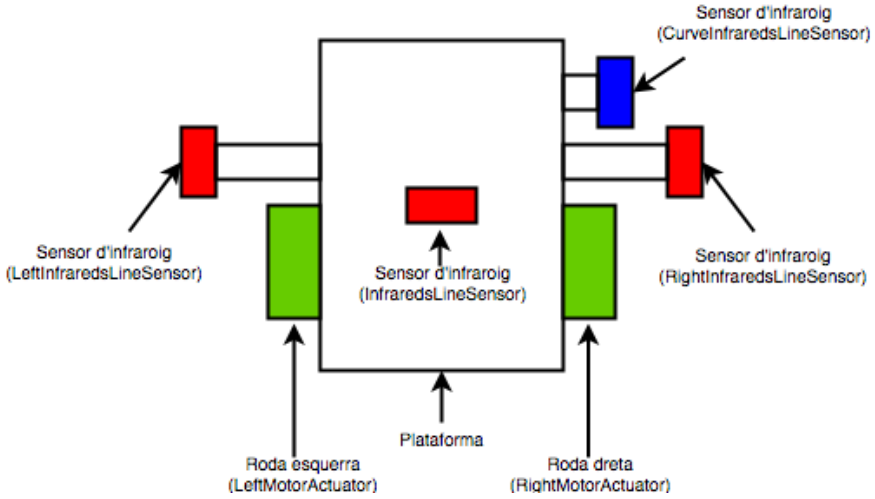

Figura 3.6: Estructura del vehicle per a la prova de la carrera de velocitat

3.5 Espai d'adaptació

Tant en la subsecció 3.3.1 i la subsecció 3.3.2 hi ha que tenir en compte la fallada dels sensors i el canvi d'estratègia quan es donen esdeveniments que identifiquen una situació que requereix que el vehicle s'adapte.

Les figures 3.8 i 3.7 mostren els diagrames de flux on es defineixen tots els possibles estats en els quals pot estar el vehicle en cada prova i quins són els events que podrien donar lloc a l'adaptació del vehicle i provocar un canvi en l'estratègia o un canvi de sensor que permeta seguir obtenint les dades necessàries.

Com es pot observar, una gran quantitat de nombres d'estats i transicions que identifiquen els events conformen els diagrames de flux. Per al cas del diagrama de la resolució del laberint són un total de 24 estats i 84 transicions, en canvi el diagrama de la carrera de velocitat és més simple, inclús així 8 estats i 20 transicions conformen el diagrama. Amb aquest nombre d'estats i de transicions no és viable el disseny i el desenvolupament d'un programari *convencional* ja que la complexitat és molt alta.

Aleshores, per al disseny i el desenvolupament de programari amb aquest nivell de complexitat és més viable el desenvolupament d'un programari

adaptable on especificant els canvis que hi ha que realitzar quan un component canvia el seu estat i amb l'ajuda del bucle *MAPE-K* és dona lloc a la solució.

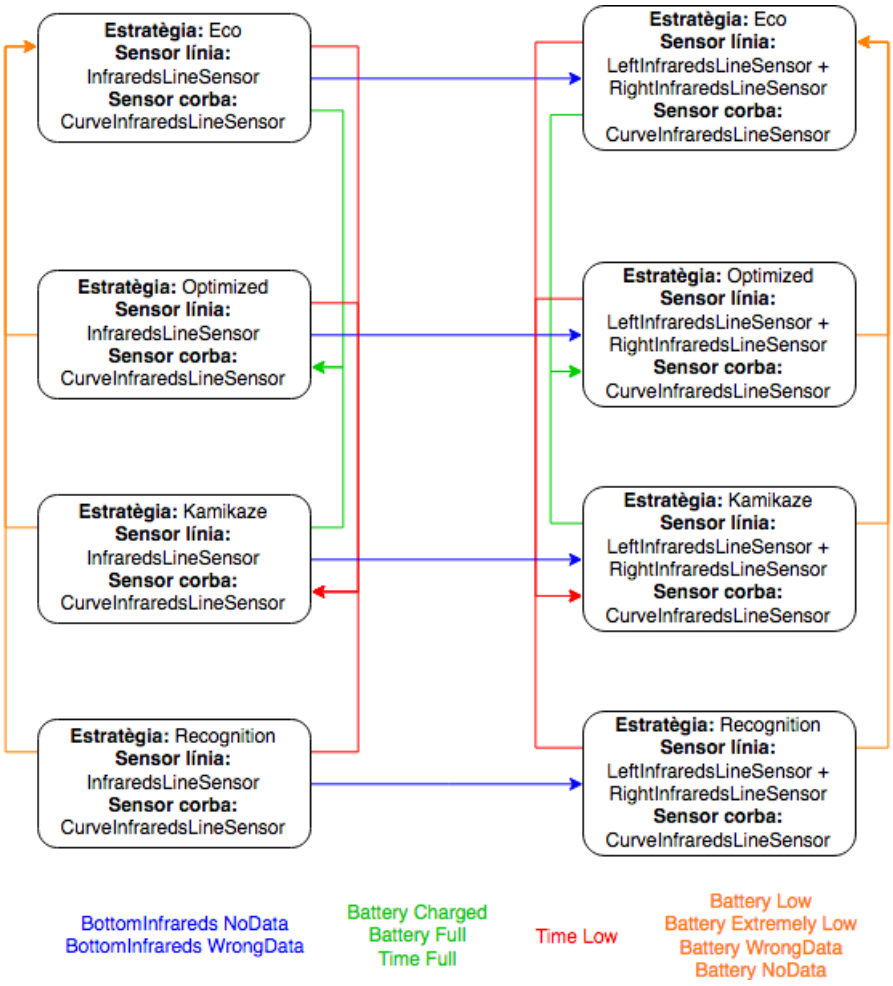

Figura 3.7: Diagrama de flux per a la carrera de velocitat

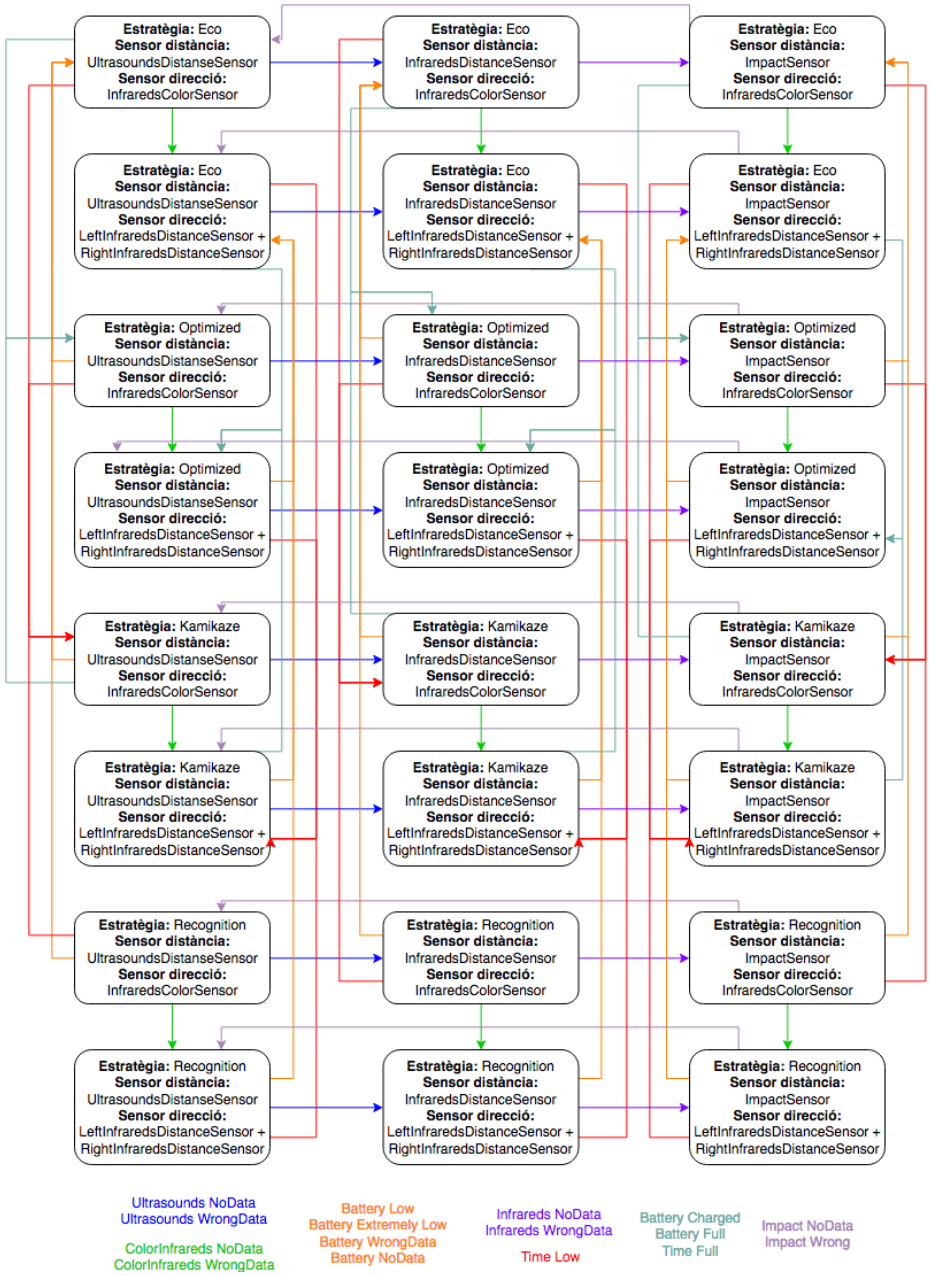

Figura 3.8: Diagrama de flux per a la resolució del laberint

Capítol 4

Computació autònoma

En aquest capítol s'introdueix un poc el món sobre la computació autònoma, així com l'explicació de la solució autoadapatativa utilitzada i les seves fases.

4.1 Introducció

La computació autònoma és aquella que ens permet definir un sistema capaç d'executar-se sense la necessitat de la intervenció humana i que facilite la implementació de sistemes grans i complexos. Aquella computació que és capaç d'adaptar-se automàticament front a esdeveniments ocorreguts segons el seu estat actual.

A més, la computació autònoma ha de ser capaç de detectar i corregir fallades per a que la seva disponibilitat no decaiga i així garantitze el seu funcionament. Com té disponible el seu estat deu garantir un funcionament òptim dels seus components, així com protegir-se d'atacs que intenten perjudicar la seva integritat i la seva seguretat.

4.2 Teoria de control

La teoria de control explica com un sistema de control deu actuar front a unes entrades per a així aplicar unes accions que impliquen canvis i s'aproximen a un objectiu. En el cas dels sistemes industrials, aquests sistemes són utilitzats primer per a obtenir i després mantindre l'estat d'una certa variable com puga ser la temperatura, la pressió o la velocitat d'un sistema. En l'àmbit que afecta a aquest projecte, el sistema de control té la funcionalitat de permitir l'adaptabilitat del vehicle per a que aquest alcance l'estat objectiu quan siga necessària la seva adaptació.

4.3 Bucles MAPE-K

Una de les solucions disponibles per a una computació autònoma és aquella que va proposar *IBM* denominada *Bucles MAPE-K*. Aquesta solució consisteix en l'execució de 4/5 fases, on cadascuna d'elles executa unes tasques específiques i aporten informació per a la fase posterior. La quinta fase (coneixement) podria no rebre la descripció de *fase* ja que es tracta d'emmagatzemament d'informació que poden consultar les altres fases, no realitza ninguna execució. En la figura 4.1 es mostra l'estructura mencionada.

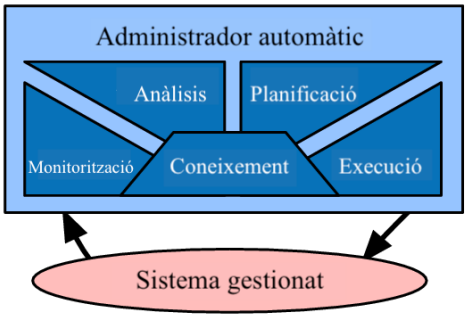

Figura 4.1: Estructura bucle MAPE-K

4.3.1 Monitorització

Primera fase del bucle MAPE-K que permet l'obtenció i la filtració dels esdeveniments generats per les sondes. Una vegada analitzada la informació obtinguda, propaga l'esdeveniment corresponent cap a la fase d'anàlisis.

4.3.2 Anàlisis

Una vegada rebut l'esdeveniment per part de la fase de monitorització, aquesta fase compara l'estat actual del sistema amb el l'estat objectiu per a actuar en conseqüència. Aquesta fase actua mitjançant regles d'adaptació, les quals especifiquen els components involucrats.

4.3.3 Planificació

Fase de presa de decisions respecte als components relacionats amb les regles d'adaptació de la fase d'anàlisis la qual permet definir quins elements deuen canviar el seu estat per a poder complir amb l'estat objectiu.

4.3.4 Execució

Última fase del bucle MAPE-K on una vegada planificades les accions necessàries per a alcançar l'estat objectiu són portades a terme. Aquestes accions són executades en aquesta fase, on finalment el sistema gestionat realitza els canvis per a satisfer l'estat objectiu.

4.3.5 Coneixement

Component que emmagatzema l'estat del sistema amb el qual es coneix la disponibilitat dels sensors, dels actuadors o quina és l'acció que està realitzant actualment el sistema per a poder pendre decisions.

Capítol 5

Anàlisis del problema

En aquest capítol s'exposa el model inicial creat que representa el vehicle a partir del qual es creen els components que permeten l'adaptabilitat. A més s'identifiquen quins són els events que es volen tindre en compte per a que el sistema s'adapte i quines funcions poden donar lloc a aquests events.

5.1 Introducció

Per a poder obtenir una solució al projecte, primerament és necessari la realització d'un plantejament inicial per a disposar els elements disponibles, com puguen ser els components que conformen el vehicle i la relació entre ells o les adaptacions que es volen tindre en compte.

5.2 Model conceptual del vehicle

Per a poder planificar un sistema adaptable, inicialment hi ha que disposar d'un model que ens permeta visualitzar un esquema global. La figura 5.1 mostra aquest model plantejat que permet identificar els components i les relacions que són trobades en el sistema. A més ens permet identificar quins elements poden ser reemplaçats.

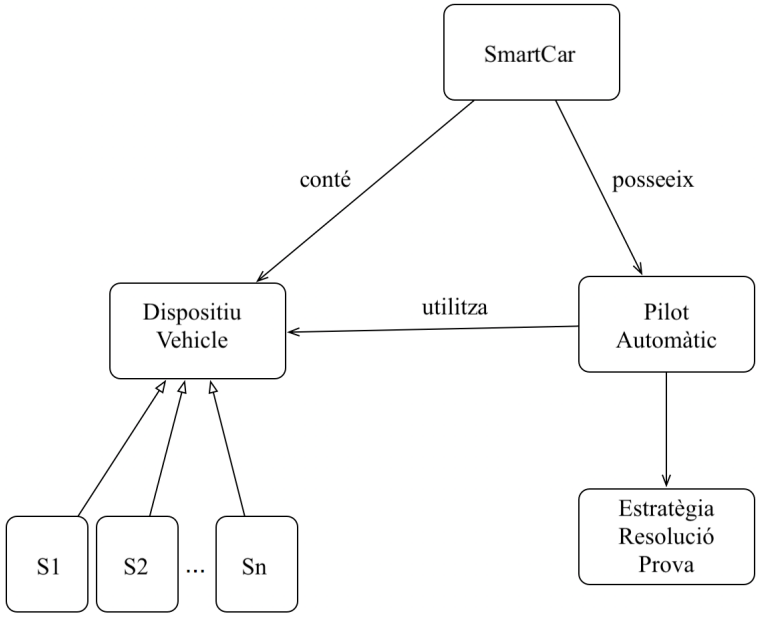

Figura 5.1: Relació entre els components del vehicle

El model està dividit en tres parts relacionades entre sí. Per una part es troba el node vehicle el qual està compost per altres nodes més bàsics com son els sensors i els actuadors necessaris. El següent node que es pot identificar es aquell que permet una resolució autònoma de les proves mitjançant les estratègies desenvolupades que el componen. Per últim es troba el node global el qual integra els dos nodes anteriors i permet el seu funcionament conjuntament.

5.3 Capacitats de computació autònoma

5.3.1 Identificació d'adaptacions

Per a poder realitzar el disseny d'un programari adaptable, primerament és necessari identificar quins tipus d'adaptacions són possibles. En aquest cas els casos donats poden ser classificats en dos grups: events i funcions. El

primer grup involucra a aquells events que poden donar-se amb la lectura dels valors dels sensors i actuar en conseqüència. En canvi, en el segon grup s'identifiquen quines funcions o funcionalitats es veuen afectades quan ocorre un dels events del primer grup.

5.3.2 Identificació d'events d'adaptació

Quan el sistema canvia la seva configuració (s'adapta) és perquè algun esdeveniment no desitjat ha ocorregut. En aquest projecte els events que emeten els esdeveniments els poden provocar quatre factors.

- **Lectures errònies:** El sistema detecta que un sensor proporciona lectures anòmales a les que deuria proporcionar degut a un problema amb aquest i intenta substituir-lo per un altre amb la mateixa funcionalitat.
- **Lectures no rebudes:** El sistema no rep lectures del sensor degut a que aquest no es troba disponible, com per exemple perquè ha sigut extret i intenta substituir-lo per un altre amb la mateixa funcionalitat.
- **Nivell de bateria:** El sistema detecta que el nivell de bateria està per baix o per dalt d'un llindar i actua amb conseqüència, canviant a una estratègia més convenient.
- **Temps de la prova:** El sistema és conscient del temps restant per a completar la prova i canvia la seva configuració quan aquest temps està per baix d'un límit i així adoptar una estratègia diferent per a que puga ser finalitzada.

5.3.3 Identificació de funcions que poden ser adaptades

Segons els events que apareixen en la secció 5.3.2, dos funcions són les que poden ser adaptades.

- **Sensor:** Quan un sensor proporciona lectures errònies o no en proporciona, la seva funcionalitat deu ser proporcionada per un altre sensor. Per un sensor que es troba en la recambra podríem dir. Aquests sensors, tant el que va a ser extret del sistema com aquell que va a ser incorporat han d'oferir la mateixa funcionalitat, ja que no té ningun

sentit el reemplaçament d'un sensor per un altre que no aporta la mateixa informació.

- **Estratègia:** El canvi d'estratègia deu ser dut a terme quan el nivell de bateria o el temps restant d'una de les proves siga superior o inferior a uns límits establerts. Per exemple, si el nivell de bateria és baix, una estratègia *eco* deuria ser activada per que tinga lloc un consum de bateria inferior i així poder acabar de realitzar la prova. O a la inversa, quan el temps restant per a la finalització d'una prova és baix es deuria activar una estratègia que fera funcionar el vehicle el més ràpid possible.

Capítol 6

Disseny de la solució

En aquest capítol s'exposen les estratègies pensades per a la resolució de les proves. A més es mostren tots els components (ARCs) involucrats en la solució i el resultat d'aplicar la solució MAPE-K en el projecte.

6.1 Estratègies

Les estratègies que es descriuen a continuació han sigut pensades segons el temps i el nivell de bateria restants que afecten al vehicle. Com que el programari desenvolupat està pensant per executar dos de les proves de l'ORC, ha sigut necessària la replicació de les estratègies per a les dos proves. Per tant les estratègies pensades són:

- **KamikazeLabyrinthStrategy:** Estratègia pensada per a que el vehicle funcione amb la velocitat màxima i arrisque el màxim possible al girar per l'interior del laberint. Aquesta estratègia es activada quan el temps per a la finalització de la prova està apunt de vèncer.
- **OptimizedLabyrinthStrategy:** Estratègia pensada per a que el vehicle funcione amb una velocitat mitjana al recórrer el laberint.
- **EcoLabyrinthStrategy:** Estratègia pensada per a que el vehicle funcione amb una limitació de velocitat per a que el consum de bateria siga el menor possible. Aquesta estratègia es activada quan el sensor que monitoritza la bateria detecta que queda un nivell inferior a un

llindar establert o per la fallada d'algun sensor no reemplaçable que impedisca el funcionament normal del vehicle al recórrer el laberint.

- **RecognitionLineFollowerStrategy:** Estratègia pensada per a que el vehicle aprenga el mapa del laberint i així poder aplicar una estratègia més agressiva posteriorment. Aquesta estratègia es activada a l'inici de la prova.

- **KamikazeLineFollowerStrategy:** Estratègia pensada per a que el vehicle funcione amb la velocitat màxima i arriesgue el màxim possible al girar les corbes del circuit arcat. Aquesta estratègia es activada quan el el temps per a la finalització de la prova està apunt de vèncer.

- **OptimizedLineFollowerStrategy:** Estratègia pensada per a que el vehicle funcione amb una velocitat mitjana al recórrer el circuit.

- **EcoLineFollowerStrategy:** Estratègia pensada per a que el vehicle funcione amb una limitació de velocitat per a que el consum de bateria siga el menor possible. Aquesta estratègia es activada quan el sensor que monitoritza la bateria detecta que queda un nivell inferior a un llindar establert o per la fallada d'algun sensor no reemplaçable que impedisca el funcionament normal del vehicle al recórrer el circuit.

- **RecognitionLineFollowerStrategy:** Estratègia pensada per a que el vehicle aprenga el mapa del circuit i així poder aplicar una estratègia més agressiva posteriorment i optar a utilitzar les dreceres disponibles en el circuit. Aquesta estratègia es activada a l'inici de la prova.

6.2 Modularització

El programari desenvolupat per al projecte pot ser dividit en dues parts.

Per una banda es troba la part funcional, on és implementada la resolució de les proves mitjançant les estratègies descrites, obtenir els valor dels sensors o actuar sobre els motors per a que el vehicle es desplace. Aquesta part ha sigut desenvolupada seguint paradigmes de programació clàssics, encara que no és l'objectiu del projecte.

Per una altra banda es troba la part que permet l'adaptabilitat del vehicle mitjançant l'especificació de components, monitors, regles d'adaptació

i escenaris. El mode en el que es desenvolupada aquesta part ens permet definir i incorporar un dels quatre components mencionats en qualsevol moment, no cal que el programari siga reiniciat per a que es done compte dels canvis introduïts.

6.3 Diagrames de disseny

Les següents figures mostren els grups de components utilitzats per a donar lloc a la solució. Després de cada solució hi ha una taula la qual mostra informació sobre cada component, com és la funcionalitat que aquest ofereix o d'aquelles que requereix per a funcionar. Es pot comprovar com components diferents ofereixen la mateixa funcionalitat, tal i com s'ha nombrat en la secció 3.4.

La figura 6.1 mostra que és possible dos configuracions. La primera d'elles seria aquella per la qual el vehicle podria ser manipulat manualment a través d'algún dispositiu extern. La segona (la utilitzada en aquest projecte) permet que el vehicle actue segons les estratègies implementades de manera autònoma.

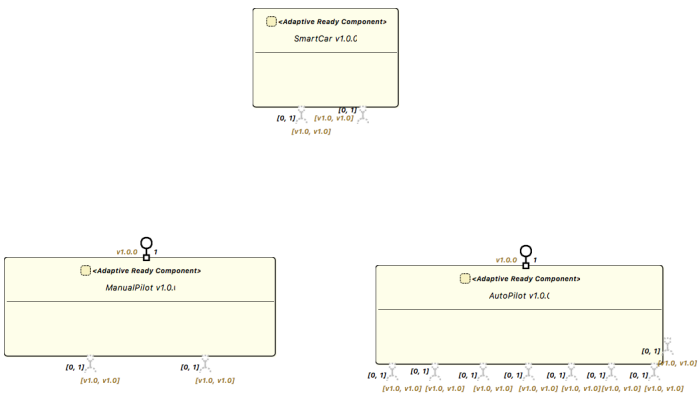

Figura 6.1: Esquema dels components del control automàtic/manual

Funcionalitat		
Component	**Ofereix**	**Requereix**
SmartCar		IAutoControl IManualControl
AutoPilot	IAutoControl	IAutoPilotStrategy IDistance IImpact IBattery ISteering IColor ILine ICurve
ManualPilot	IManualControl	ICommunication ISteering

Taula 6.1: Funcionalitats dels components del control automàtic/manual

La figura 6.2 mostra el conjunt de sensors utilitzats per a la resolució de les proves implicades. Si dividim els components de la figura en tres grups, tindríem aquells que utilitza la resolució del laberint a l'esquerra, els que utilitza la carrera de velocitat a la dreta i el sensor de bateria que es comú a les dos proves.

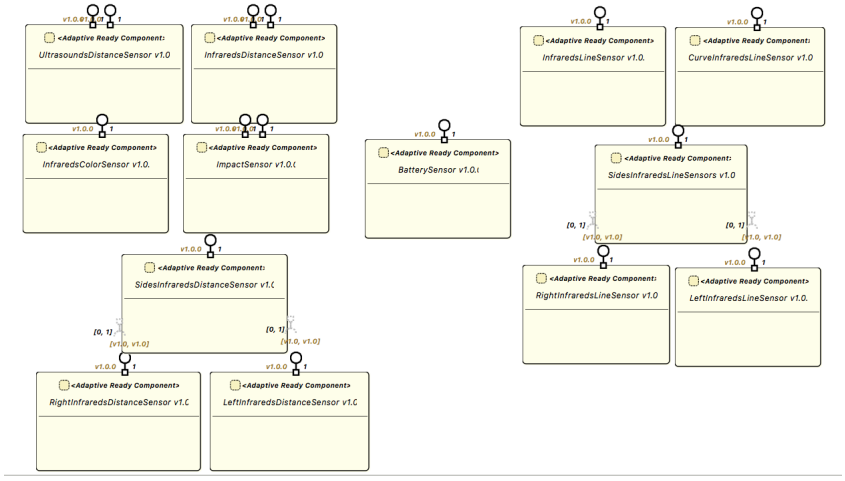

Figura 6.2: Esquema dels components de sensorització

	Funcionalitat	
Component	**Ofereix**	**Requereix**
UltrasoundsDistanceSensor InfraredsDistanceSensor ImpactDistanceSensor	IDistance IImpact	
InfraredsColorSensor	IColor	
SidesInfraredsDistanceSensor	IColor	IRightDistance ILeftDistance
RightInfraredsDistanceSensor	IRightDistance	
LefttInfraredsDistanceSensor	ILefttDistance	
BatterySensor	IBattery	
InfraredsLineSensor	ILine	
CurveInfraredsLineSensor	ICurve	
SidesInfraredsLineSensor	ILine	IRightLine ILeftLine
RightInfraredsLineSensor	IRightLine	
LeftInfraredsLineSensor	ILeftLine	

Taula 6.2: Funcionalitats dels components de sensorització

Les figures 6.3 i 6.4 representen les estratègies per a cadascuna de les proves tal i com són exposades en la secció 6.1.

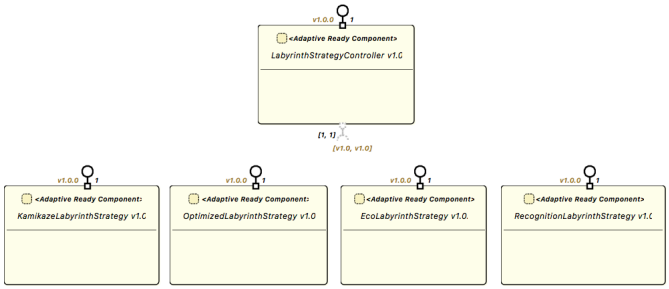

Figura 6.3: Esquema de les estratègies per al laberint

	Funcionalitat	
Component	**Ofereix**	**Requereix**
LabyrinthStrategyController	IAutoPilotStrategy	ILabyrinthStrategy
KamikazeLabyrinthStrategy OptimizedLabyrinthStrategy EcoLabyrinthStrategy RecognitionLabyrinthStrategy	ILabyrinthStartegy	

Taula 6.3: Funcionalitats de les estratègies per al laberint

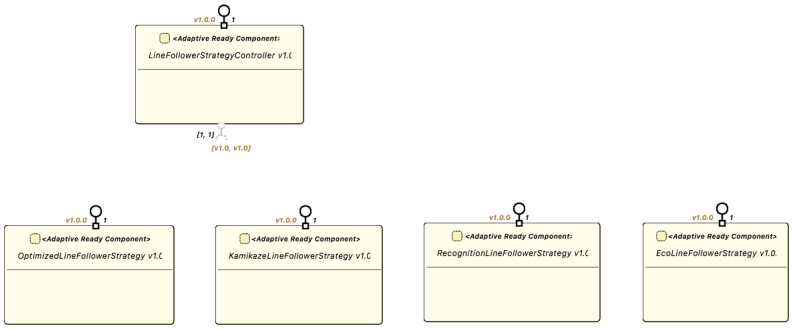

Figura 6.4: Esquema de les estratègies per a la carrera de velocitat

	Funcionalitat	
Component	**Ofereix**	**Requereix**
LineFollowerStrategyController	IAutoPilotStrategy	ILineFollower-Strategy
KamikazeLineFollowerStrategy OptimizedLineFollowerStrategy EcoLineFollowerStrategy RecognitionLineFollowerStrategy	ILineFollower-Startegy	

Taula 6.4: Funcionalitats de les estratègies per a la carrera de velocitat

La figura 6.5 exposa el conjunt d'actuadors per a permetre la mobilitat del vehicle. Aquests actuadors son dos motors, els quals segons la seva velocitat permeten anar cap endavant, cap enrere i girar cap als dos costats. Com es pot veure no es necessari un tercer motor per girar, encara que eixa podria ser una alternativa.

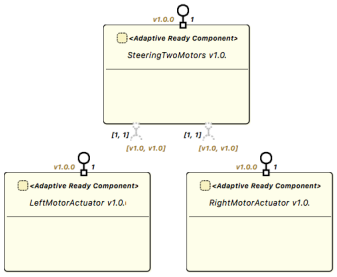

Figura 6.5: Esquema dels components de direcció

	Funcionalitat	
Component	**Ofereix**	**Requereix**
SteeringTwoMotors	ISteering	IRightEngine ILeftEngine
RightMotorActuator	IRightEngine	
LeftMotorActuator	ILeftEngine	

Taula 6.5: Funcionalitats dels components de direcció

Per últim, la figura 6.6 mostra els components mitjançant els qual el vehicle podria ser manipulat manualment. En aquest cas el vehicle podria ser manipulat mitjançant tres tecnologies diferents: *Bluetooth*, *Wi-Fi* i *Radio*.

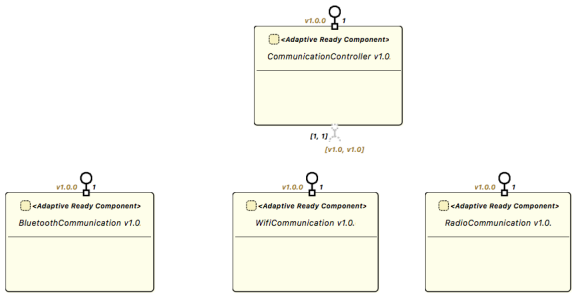

Figura 6.6: Esquema dels components de comunicació

	Funcionalitat	
Component	**Ofereix**	**Requereix**
CommunicationController	ICommunication	IController-Communication
BluetoothCommunication WifiCommunication RadioCommunication	IController-Communication	

Taula 6.6: Funcionalitats dels components de comunicació

6.4 Disseny solució emprant bucle MAPE-K

La figura 6.7 mostra l'estructura implementada per a la solució desenvolupada. La figura mostra com les fases són dividides en grups, permetentnos executar cadascuna d'elles en una màquina diferent. El component *Managed System* és el que s'executa sobre el vehicle que realitza les proves.

La comunicació entre el vehicle i les fases de monitorització i anàlisis es realitza mitjançant el direccionament *IP*, per tant, si aquestes fases s'executen sobre la mateixa màquina (el vehicle en aquest cas) la direcció *IP* seria *localhost*. En canvi si s'executa en una màquina externa, la direcció *IP* seria aquella que identifica a la màquina externa. En referència a la comunicació entre els altres grups s'utilitza un sistema de cues, concretament *MQTT* (explicat en la subsecció 2.2.1). L'element *Probes* permet configurar l'escenari inicial al arrancar el sistema.

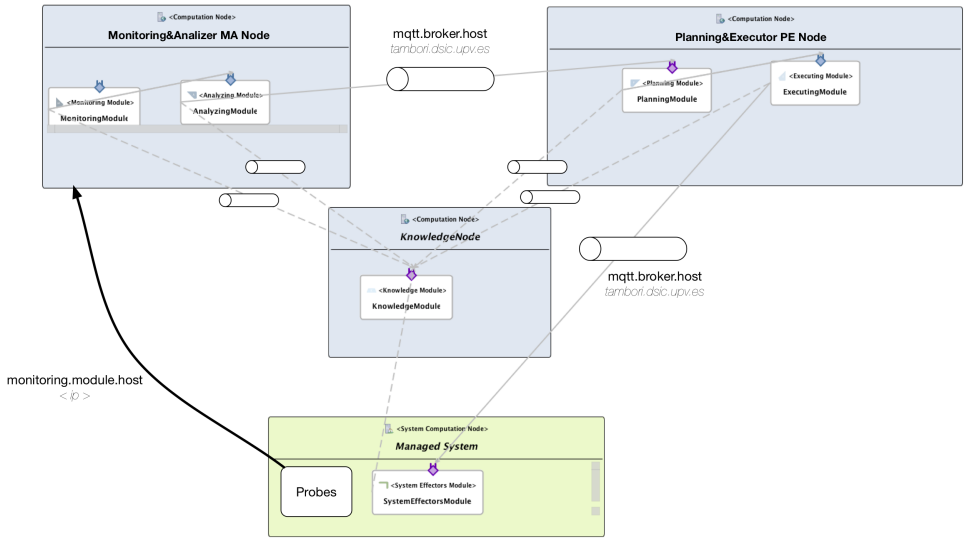

Figura 6.7: Estructura del bucle MAPE-K implementat

Capítol 7

Implementació

En aquest capítol es mostren els components desenvolupats de manera gràfica si existeix la possibilitat o fragments de codi que permeten la seva manipulació.

7.1 Components desenvolupats

Per a la realització d'un vehicle adaptable a les situacions ha sigut necessària la creació d'un programari basat en uns components que permeten aquesta adaptabilitat. En concret quatre tipus de components han sigut desenvolupats.

El primer d'ells es denominat ARC (*Adaptative Ready Component*), és aquell que permet la seva incorporació i extracció del model de components. Un segon component és aquell denominat com a regla d'adaptació (*Adaptative Rule*) que permet definir un escenari objectiu quan un esdeveniment és donat, com per exemple la detecció d'un nivell baix de bateria. Seguidament un component anomenat monitor el qual captura esdeveniments generats per les sondes i en funció d'aquests notifica esdeveniments per a que les regles d'adaptació puguen ser dutes a terme. Per últim, el quart component és el denominat com a sonda, el qual llegeix la informació provinent dels sensors i propaga esdeveniments segons el valor rebut.

7.1.1 *Adaptative Ready Component*

La figura 7.1 mostra l'estructura d'un component ARC. En ella es pot veure com un ARC té associats diversos elements, aquests elements són nomenats *Service Requirement* i *Service Supply*. L'element *Service Requirement* (color blau) permet definir quins components requereix el ARC per al seu funcionament, en canvi l'element *Service Supply* (color roig) permet definir la funcionalitat que el component ofereix als altres components del diagrama.

Per exemple, el component de la figura 7.1 ofereix la funcionalitat *IAutoControl* que serà utilitzada per un altre component (*SmartCar*). D'altra banda, requereix de 8 components distints entre els quals es troba aquell que defineix l'estratègia de resolució de les proves (*IAutoPilotStrategy*). El nombre que es mostrat al costat de cada element defineix la quantitat d'instàncies d'aquest elements que ofereix/requereix.

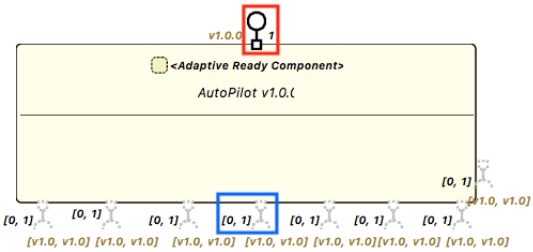

Figura 7.1: Component ARC

7.1.2 Regla d'adaptació

La creació d'una regla d'adaptació no és realitzada a través d'un entorn gràfic al igual que un ARC, és realitzada a través de codi amb el llenguatge *Java*. Per a que aquest component realitze la seva funcionalitat, és necessari la definició dels esdeveniments el qual ha d'entendre i l'acció que ha de realitzar quan aquests esdeveniments ocorren.

Per a una millor comprensió d'aquest component, en el codi 7.1 es mostra el fragment més important del codi d'un d'ells.

Aquest fragment de codi és executat quan un esdeveniment es produeix, aleshores es compara l'esdeveniment rebut amb els que són relatius a la regla. Si l'esdeveniment rebut és relatiu a la regla, la part interna de la sentència *if* serà executada en la qual es comprova quina de les dues proves està realitzant el vehicle (cc.getId().equalsIgnoreCase("LineFollowerStrategyController") o cc.getId().equalsIgnoreCase("LabyrinthStrategyController")) i associa l'escenari objectiu (nextSCmodelName = "InLineFollowerMode_OnBatteryLowLevel.systemconfigurationmodel" o nextSCmodelName = "InLineLabyrinthMode_OnBatteryLowLevel.systemconfigurationmodel") per a que els canvis necessaris siguen realitzats. És necessari saber quina prova és la que està realitzant-se en el moment en el que es dona l'esdeveniment ja que els canvis per a l'adaptació no són iguals. És possible aquesta comprovació gràcies al component coneixement del bucle MAPE-K.

```
1  if(changeEvent.getRT().equals(IChangeEvent.
       CHANGE_EVENT_TOPIC_HEADING+"Battery_Low") || changeEvent
       .getRT().equals(IChangeEvent.CHANGE_EVENT_TOPIC_HEADING+
       "Battery_Extremely_Low")) {
2      ArrayList<ISystemConfiguration>systemConfigurations =
           new ArrayList<ISystemConfiguration>();
3      try {
4          String nextSCmodelName = null;
5          SerializedSystemConfigurationModel
               current_scmSerialized = ((IKnowledge)this.
               getAdaptationFrameworkBundleContext().
               getAnalyzingModule().getKnowledgeModuleMediator
               ()).getCurrentSystemConfiguration(modelName+".
               componentsmodel");
6          SystemConfigurationModel currentConfiguration =
               ModelUtils.
               loadSystemConfigurationModelSerialized(
               current_scmSerialized, this.
               getAdaptationFrameworkBundleContext().
               getOSGiBundleContext());
7
8          if (currentConfiguration.getComponentConfigurations
               () != null) {
9              for (ComponentConfiguration cc :
                   currentConfiguration.
                   getComponentConfigurations()) {
10                 if (cc.getId().equalsIgnoreCase("
                       LineFollowerStrategyController")) {
11                     nextSCmodelName = "InLineFollowerMode_"+
12                     "OnBatteryLowLevel"+
13                     ".systemconfigurationmodel";
```

```
14                      break;
15                  } else if (cc.getId().equalsIgnoreCase("
                        LabyrinthStrategyController")) {
16                      nextSCmodelName = "InLabyrinthMode_"+
17                      "OnBatteryLowLevel"+
18                      ".systemconfigurationmodel";
19                      break;
20                  }
21              }
22          }
23
24          if (nextSCmodelName == null) {
25              super.getAdaptationFrameworkBundleContext().
                    getRegisteredAdaptationDecider().
                    setRuleAsNotAffectedByChangeEvent(
                    changeEvent, super.getID());
26              return;
27          }
28          SerializedSystemConfigurationModel
                next_scmSerialized = ((IKnowledge)this.
                getAdaptationFrameworkBundleContext().
                getAnalyzingModule().getKnowledgeModuleMediator
                ()).loadSystemConfigurationModel(nextSCmodelName
                );
29          SystemConfigurationModel nextConfiguration =
                ModelUtils.
                loadSystemConfigurationModelSerialized(
                next_scmSerialized, this.
                getAdaptationFrameworkBundleContext().
                getOSGiBundleContext());
30          AF_ModelBasedSystemConfiguration mbsc= new
                AF_ModelBasedSystemConfiguration(
                nextConfiguration);
31
32          systemConfigurations.add(mbsc);
33      }
34      catch (Exception e) {
35          e.printStackTrace();
36      }
37      super.getAdaptationFrameworkBundleContext().
            getRegisteredAdaptationDecider().
            setRuleAsAffectedByChangeEvent(changeEvent, super.
            getID(), systemConfigurations);
38  }
```

Codi 7.1: Associació de l'escenari objectiu

La figura 7.2 mostra l'escenari objectiu quan s'ha donat un nivell de bateria baix i el vehicle es troba realitzant la resolució del laberint. El color roig als components significa que aquests deuen ser parats ja que un altre component amb la mateixa funcionalitat va a ser executat. En la figura es paren la resta d'estratègies i s'activa l'estratègia *eco*.

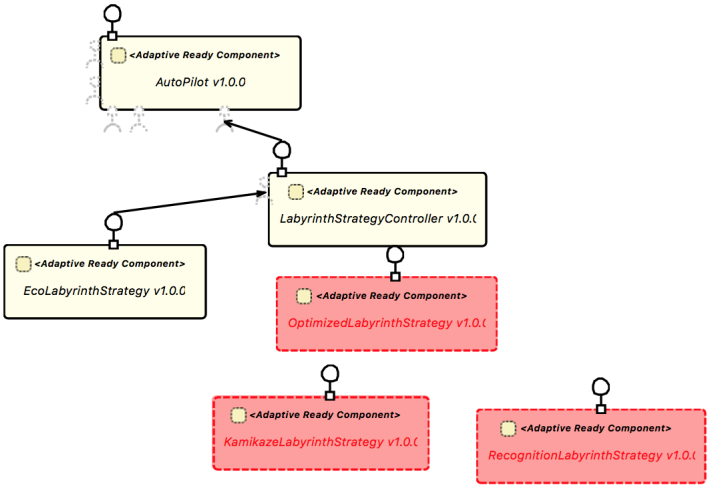

Figura 7.2: Escenari objectiu

7.1.3 Monitor

Al igual que els components denominats regles d'adaptació, els monitors que obtenen els valors de les sondes són creats mitjançant codi amb el llenguatge *Java*. Aquest component permet l'enviament d'esdeveniments que són capturats per les regles d'adaptació tal i com s'ha vist en el punt anterior.

Com es pot veure en el codi 7.2, segons el valor rebut (*Battery_Low*, *Battery_Extremely_Low*, etc.) s'associa un esdeveniment o altre (outcomingRT), el qual és enviat posteriorment (this.throwChangeEvent(event)).

```
1 String outcomingRT = null;
2 if (incomingRT.equalsIgnoreCase("Battery_Low")) {
3     outcomingRT = BatteryMonitor_Monitor.CHANGE_EVENT_RT_3;
4 } else if (incomingRT.equalsIgnoreCase("
    Battery_Extremely_Low")) {
```

```
5       outcomingRT = BatteryMonitor_Monitor.CHANGE_EVENT_RT_4;
6   } else if (incomingRT.equalsIgnoreCase("Battery_Charged")) {
7       outcomingRT = BatteryMonitor_Monitor.CHANGE_EVENT_RT;
8   } else if (incomingRT.equalsIgnoreCase("Battery_Full")) {
9       outcomingRT = BatteryMonitor_Monitor.CHANGE_EVENT_RT_2;
10  } else if (incomingRT.equalsIgnoreCase("Battery_WrongData"))
        {
11      outcomingRT = BatteryMonitor_Monitor.CHANGE_EVENT_RT_5;
12  } else if (incomingRT.equalsIgnoreCase("Battery_NoData")) {
13      outcomingRT = BatteryMonitor_Monitor.CHANGE_EVENT_RT_6;
14  } else {
15      return;
16  }
17
18  AF_ChangeEvent event = new AF_ChangeEvent(outcomingRT);
19
20  this.throwChangeEvent(event);
```

Codi 7.2: Enviament d'un esdeveniments

7.1.4 Sonda

El component sonda ens permet decidir quin esdeveniment (RT) enviar (this.send(event)) cap als monitors segons els valors rebuts dels sensors que composen el vehicle. En el codi 7.3 s'obté el nivell de bateria, s'emmagatzema en la variable *level* i es decideix quin esdeveniment enviar segons el seu valor. Per a les proves inicials convé que les sondes siguen simulades fixant els valors de les variables. Una vegada desenvolupat tot el programari o gran part d'ell ja és recomanable l'obtenció del valors a través dels dels sensors reals.

```
1   String RT = null;
2   if (level >= 100)
3       RT = "Battery_Full";
4   else if (level > 80)
5       RT = "Battery_Charged";
6   else if (level < 5)
7       RT = "Battery_Extremely_Low";
8   else if (level < 20)
9       RT = "Battery_Low";
10  else
11      return;
12
13  ISystemEvent event = Probe.createEvent(RT, null);
```

```
14  this.send(event);
```

Codi 7.3: Enviament d'un esdeveniment segons el nivell de bateria

7.2 Execució grups bucle MAPE-K

Tal i com s'ha explicat en la secció 6.4, les fases d'aquest tipus de solució poden ser dividides en grups i col·locades en diferents màquines. Les figures següents mostren com cadascun d'aquests grups està situat en una *Raspberry Pi* diferent.

Per a donar cabuda a aquesta solució cinc *Raspberry Pi*s són necessàries, una per al node/grup *MA* que composa les fases de monitorització i anàlisis, un altra per al node/grup *PE* que inclueix les fases de preparació i execució i una tercera per al node que conté el coneixement (K) de l'estat del sistema. Aquests tres dispositius componen les fases del bucle *MAPE-K* i els altres dos restants contenen el sistema gestionat (el vehicle en aquest cas) i el *broker MQTT* per a la retransmissió de missatges tal i com s'explica en la subsecció 2.2.1.

La figura 7.3 mostra les eixides proporcionades pels dispositius quan aquests són arrancats a excepció del *broker MQTT*. D'aquesta figura es pot traure informació respecte a les connexions que tenen els grups entre ells. Per exemple, el sistema gestionat (finestra de dalt a l'esquerra) mostra la connexió amb el *broker MQTT* (*Connected to tcp://192.168.5.1:1883*).

En canvi, la figura 7.4 reflecteix les eixides dels grups quan s'inicialitza el sistema amb l'escenari inicial. Es pot comprovar com el sistema gestionat (finestra de dalt a l'esquerra) realitza les instanciacions dels components i com el node/grup PE (finestra de baix a la dreta) ha indicat els canvis a realitzar per a que el sistema gestionat arribe fins a l'escenari objectiu.

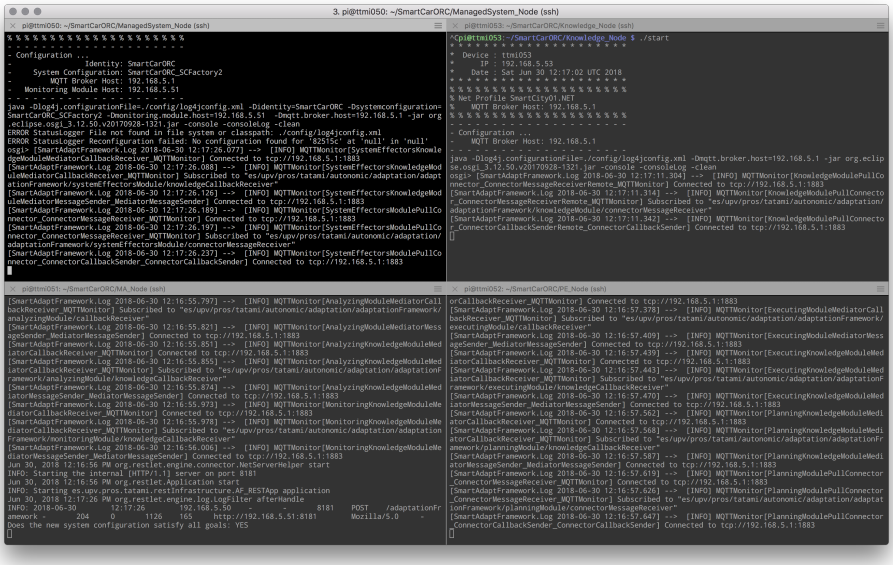

Figura 7.3: Estat inicial grups blucle MAPE-K

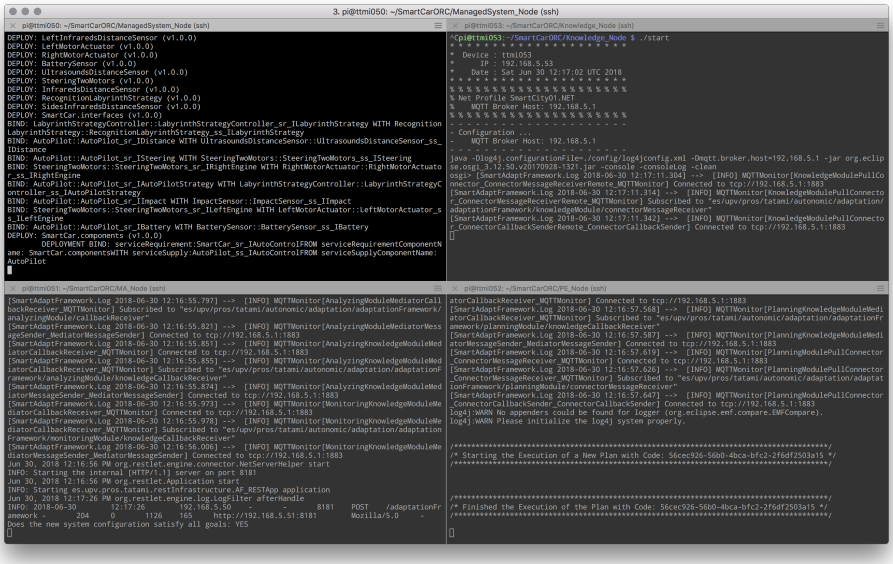

Figura 7.4: Estat canvi grups blucle MAPE-K

Capítol 8

Proves de la solució

En aquest capítol es mostra un exemple d'ús mitjançant dos diagrames de com al ocórrer esdeveniments no desitjats el vehicle canvia de sensor o d'estratègia. A més es mostra una prova realitzada on es comprova l'estat dels components abans i després de que el sensor de bateria indique un nivell de bateria baix i falle el sensor de distància.

8.1 Exemples d'ús

En les següents figures es mostren dos diagrames que representen els passos que segueix el vehicle quan ocorre un esdeveniment no desitjat. En la figura 8.1 quan l'esdeveniment *battery_low* (Bateria baixa en el diagrama) ocorre el sistema canvia a una nova estratègia, en concret es canvia qualsevol de les tres estratègies de l'esquerra a l'estratègia *eco*.

En canvi, en la figura 8.2 es canvia el sensor que detecta la distància amb la paret frontal quan un d'ells no es troba disponible. Les relacions entre ells han sigut definides per decisió pròpia, aquestes podrien definides d'un altre mode. Per exemple, el sensor d'impacte podria ser l'opció que seguira al sensor d'ultrasons i no el d'infraroig.

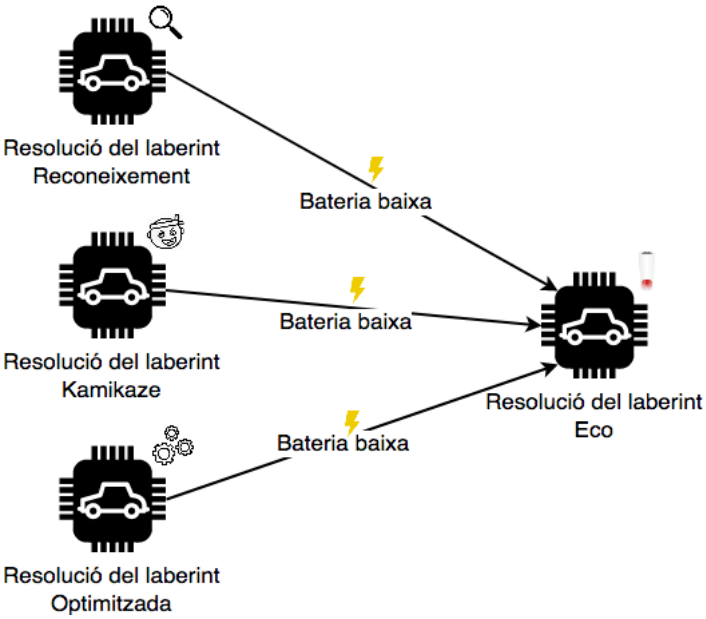

Figura 8.1: Adaptació front a un nivell baix de bateria

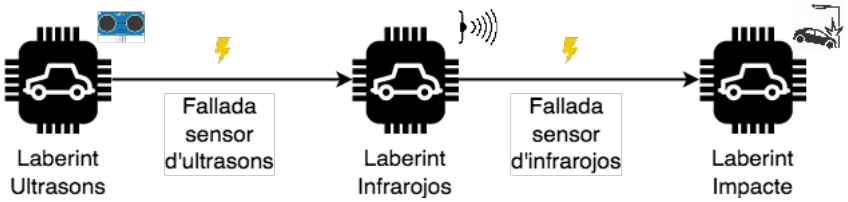

Figura 8.2: Adaptació front a fallades del sensor de distancia

8.2 Proves

Una vegada observats els diagrames de la secció 8.1, passem a comprovar que el programari desenvolupat compleix amb aquestes transicions quan ocorren els esdeveniments implicats.

Per a la comprovació del programari desenvolupat sense la necessitat de tindre un vehicle físic, un programari addicional va ser desenvolupat per a simular la funció que realitzen les sondes, és a dir, l'enviament d'esdeveni-

ments quan es dona la fallada de sensors o quan el nivell de bateria o el temps restant de la prova superen un llindar.

Quan el programari arranca, un escenari d'inici deu ser carregat en el vehicle per a que es pose en funcionament. En la figura 8.3 es mostra l'escenari inicial desenvolupat. Es pot veure com el vehicle arranca amb una configuració que realitza la prova de la resolució del laberint amb l'estratègia de reconeixement, al mateix temps que es completa amb els sensors i actuadors necessaris per a la realització de la prova.

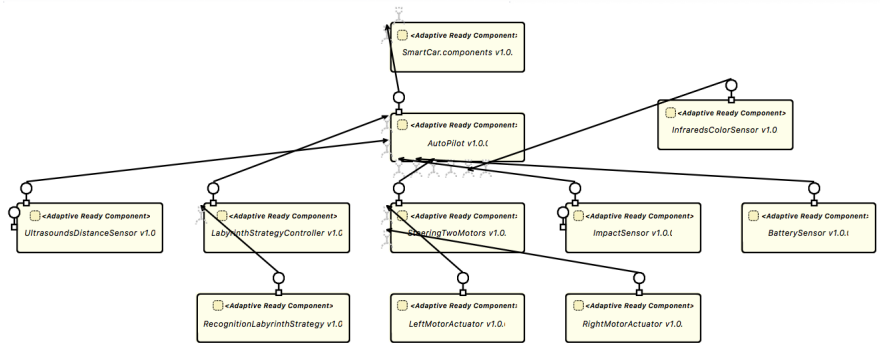

Figura 8.3: Estructura de l'escenari inicial

L'ordre *ss* que ofereix el *framework OSGi* permet veure l'estat dels components del vehicle, mostrant quins d'ells estan actius i quins d'ells no. En la figura 8.4 es mostra com els components que formen l'escenari inicial es troben tots com actius (sols es mostren els implicats perquè són un gran nombre).

```
6       ACTIVE      SteeringTwoMotors_1.0.0
12      ACTIVE      LeftMotorActuator_1.0.0
14      ACTIVE      AutoPilot_1.0.0
17      ACTIVE      BatterySensor_1.0.0
18      ACTIVE      RightMotorActuator_1.0.0
21      ACTIVE      InfraredsColorSensor_1.0.0
39      ACTIVE      SmartCar.components_1.0.0
56      ACTIVE      UltrasoundsDistanceSensor_1.0.0
71      ACTIVE      RecognitionLabyrinthStrategy_1.0.0
79      ACTIVE      LabyrinthStrategyController_1.0.0
97      ACTIVE      ImpactSensor_1.0.0
```

Figura 8.4: Estat dels components de l'escenari inicial

Una vegada comprovat que l'escenari inicial ha sigut carregat correctament, es hora de comprovar com el sistema s'adapta quan un esdeveniment no desitjat ocorre. Els diagrames de la secció 8.1 corresponen als estats de la figura 8.5, la qual representa un subconjunt de la figura 3.8.

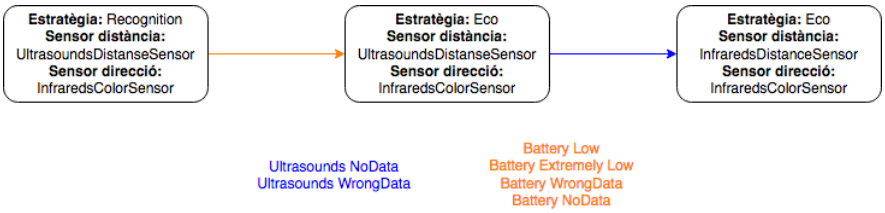

Figura 8.5: Transicions a executar

Amb l'ordre *battery* X, on X $\in [0, 100]$, es possible enviar un nivell de bateria al sistema, el qual segons aquest valor envia l'esdeveniment corresponent. En aquest exemple l'ordre utilitzada és *battery 15*, per tant s'envia un esdeveniment que indica que la bateria està a un nivell baix que és capturat pel seu monitor associat.

Una vegada executada aquesta ordre, el sistema s'adapta canviant la seva estratègia de la de reconeixement a la *eco*, la qual realitza la prova d'una manera més lenta i permet estalviar bateria. En la figura 8.6 es mostra com l'estat dels components de les estratègies ha sigut canviat (sols es mostren els components afectats).

```
71      RESOLVED    RecognitionLabyrinthStrategy_1.0.0
77      ACTIVE      EcoLabyrinthStrategy_1.0.0
```

Figura 8.6: Estat dels components després de *battery 15*

Una vegada canviada l'estratègia de resolució del laberint, amb l'ordre *ultrasound wrong* es provoca que el sensor d'ultrasons, que en eixe moment mesura la distància que hi ha amb la paret frontal, senyalitze un esdeveniment de fallada i obligue al sistema a canviar de sensor per a mesurar la distància. Per lo tant i segons el diagrama, el sensor que serà activat és el d'infraroig. L'estat dels components afectats després de l'execució d'aquesta ordre es pot veure en la figura 8.7.

```
56      RESOLVED    UltrasoundsDistanceSensor_1.0.0
11      ACTIVE      InfraredsDistanceSensor_1.0.0
```

Figura 8.7: Estat dels components després de *ultrasound wrong*

En la secció 8.3 es mostra una taula que conté totes les ordres de simulació disponibles per a provocar que el vehicle s'adapte. Hi ha tant ordres numèriques (*battery 15*) com textuals (*ultrasound wrong*).

8.3 Ordres de simulació disponibles

Ordre	**Paràmetre**	**Descripció**	**Exemples**
battery	$[0, 100]$	Permet definir un nivell de bateria restant o si hi ha	battery 15
	wrong	algun problema amb el sensor	battery wrong
	nodata	de la bateria	battery nodata
bottom-Infrared	wrong	Permet indicar si hi ha algun problema amb el sensor infraroig inferior	bottom-Infrared wrong
	nodata		bottom-Infrared nodata
color-Infrared	wrong	Permet indicar si hi ha algun problema amb el sensor infraroig frontal	color-Infrared wrong
	nodata		color-Infrared nodata
infrared	wrong	Permet indicar si hi ha algun problema amb el sensor	infrared wrong
	nodata	d'infraroig	infrared nodata
impact	wrong	Permet indicar si hi ha algun problema amb el sensor	impact wrong
	nodata	d'impacte	impact nodata
time	low	Permet indicar el temps	time low
	high	restant	time high
ultrasound	wrong	Permet indicar si hi ha algun problema amb el sensor	ultrasound wrong
	nodata	d'ultrasons	ultrasound nodata

Taula 8.1: Ordres de simulació disponibles

Capítol 9

Conclusions i treballs futurs

En aquest capítol es mostren les conclusions sobre el treball i les possibles ampliacions futures.

9.1 Conclusions

En aquest document s'ha explicat la realització del programari per al disseny de l'adaptabilitat d'un vehicle. Per a la realització d'aquest programari ha sigut necessari passar per diverses fases, des de especificar el muntatge del vehicle, decidir quins events poden ocórrer i davant els quals el vehicle deu adaptar-se i implementar aquestes idees amb el programari disponible.

Per últim i després d'haver realitzat el projecte, es pot dir que els objectius plantejats en la secció 1.3 han sigut alcançats en la seva totalitat, a més ha pogut ser comprovat amb les situacions recreades mitjançant un programari addicional que simula el comportament dels components tal i com s'ha comprovat en la secció 8.2.

9.2 Treballs futurs

Com ha sigut nomenat durant la memòria el programari desenvolupat està enfocat per a dos de les proves de l'ORC. Com a treball futur es podria dur a terme el desenvolupament del programari necessari per a la resta de les proves que componen la competició.

Un altre possible treball futur és la implementació de la manipulació manual per part de l'usuari del vehicle. En els diagrames desenvolupats es troben els components necessaris per a la comunicació, definida amb diverses tecnologies.

Per últim, la realització de les proves ha sigut realitzada a mà i sols s'ha mostrat dos possibles execucions d'ordres de simulació. Una possible millora o treball futur seria la realització de proves automàtiques per a poder abordar tots els casos i així comprovar el funcionament total del sistema.

9.3 Valoració personal

La realització d'aquest projecte m'ha permès descobrir la complexitat de les fases a seguir per a realitzar un programari. Fins ara, no m'havia parat mai a fer ninguna especificació de requisits funcionals, però donada la complexitat d'aquest projecte va ser una de les primeres coses que vaig dur a terme. Després de l'especificació va seguir el pas de plasmar-ho amb uns *frameworks* totalment nous per a mi, això va fer que haguera d'aprendre les seves característiques i les funcionalitats útils que ofereixen per a aquest projecte.

Amb l'ajuda del tutor he après moltes coses noves referents al programari, als *frameworks*, a desenvolupar programari adaptatiu (sempre havia sigut funcional) i sobre l'IDE Eclipse el qual s'utilitza bastant i estic satisfet amb els resultats obtinguts.

Bibliografia

[1] *OSGi bundles*. URL: https://www.ibm.com/support/knowledgecenter/es/SS8PJ7_9.6.1/com.ibm.aries.osgi.doc/topics/cbundles.html#bundlelifecycle (v. la pàg. 7).

[2] *FESAS*. URL: https://fesas.bwl.uni-mannheim.de/about/ (v. la pàg. 9).

[3] *Genie*. URL: http://genie-framework.sourceforge.net/#d0e42 (v. la pàg. 9).

[4] *Kevoree*. URL: http://kevoree.org/docs/getting-started/what-is-kevoree.html (v. la pàg. 10).

[5] *Kit Iniciación a la robótica*. URL: https://orchallenge.es/talleres#kit (v. la pàg. 12).

[6] Jaume Laborda. *Recursos talleres*. 2017. URL: https://drive.google.com/drive/folders/0B37SSskaN8LkN2FHRDhzdlBqZms (v. la pàg. 12).

[7] Jaume Torres Pous. *Normativa General ORC 2.17*. 2017. URL: https://drive.google.com/file/d/0B37SSskaN8LkWmpGME5wVFFqRGM/view (v. la pàg. 13).

[8] Kashif Dar. *MAPE-K adaptation control loop*. 2012. URL: http://www.uio.no/studier/emner/matnat/ifi/INF5360/v12/undervisningsmateriale/MAPE-K%20adap%20control%20loop.pdf.

[9] Francisco Presencia i Jaime Laborda. *Olympic Robotic Challenge*. URL: https://orchallenge.es/.

[10] Joan Fons i Cors. *La Computación Autónoma*.

[11] Prof. Francisco M. Gonzalez-Longatt. *Introducción a la Teoría de Control*. 2008. URL: https://www.researchgate.net/publication/296676720_Introduccion_a_la_Teoria_de_Control.