UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

TREBALL DE FI D'ESTUDIS

Estudi, disseny i implementació d'un sistema de supervisió, control i adquisició de dades usant tecnologies web per al control i monitorització d'un motor DC

Document:
Memòria

Autor:
Achraf El Yahyati Anahas

Director - Codirector:
Alejandro Bachiller – Jaume Figueras

Titulació:
Grau en Electrònica Industrial i Automàtica

Convocatòria:
Primavera, 2022

Resum:

Aquest projecte consisteix en la creació d'una pàgina web que permeti la monitorització i el control sobre un motor DC. Per portar-ho a terme, es realitzarà un estudi de quins llenguatges web són els més utilitzats en l'actualitat i quins s'adapten més a les condicions d'aquesta aplicació. També s'haurà d'analitzar el procés d'adquisició i tractament de dades entre el motor i el software escollit.

L'objectiu principal de la web serà que l'usuari li pugui assignar una consigna al motor i, posteriorment, pugui veure en una gràfica la lectura del comportament del motor. Aquest procés es podrà fer tan en llaç obert com en llaç tancat. En el primer cas, l'usuari podrà apreciar el comportament del motor sense cap tipus de control. En el segon cas, juntament amb la consigna, es podran configurar les constants del controlador PI; Kp i Ki. D'aquesta manera, es podrà sintonitzar el controlador i aconseguir la millor resposta del motor directament des de la web.

En definitiva, el projecte serà ideal per poder combinar l'àmbit de la programació o automatització amb el sector del control. Enllaçar llenguatges de programació web amb programes de modelització com podria ser Matlab serà clau per la correcta realització del projecte.

Abstract

The aim of this project is to create a website for monitoring and controlling a DC motor. For this purpose, we are going to research about the most used web languages in our days and determine which of those are more suitable for our application. We will be analyzing how the data is acquired and treated between the motor and the chosen software.

The main objective of the website is to allow the user to give an order to the motor and to see how the motor behaves by interpreting a chart. This process might be done both in open loop and closed loop. In the first case, the user may see the motor's behavior with no controlling options. In the second case, beside/apart from the Setpoint, the user may also set the PI controller constants, Kp and Ki; which allows the controller to be synchronized to reach the best response of the motor directly through the website.

In summary, this project is the perfect opportunity to mix two interesting fields. On one side, the programming and automation, and on the other, the controlling. The key to achieve a good result in this project lies in combining the programming languages with the modelling programs such as Matlab.

Índex:

Índex de figures:

Índex de taules

1 Introducció:

Objecte:

L'objectiu principal del projecte és el control i monitorització d'un motor DC des de una pàgina web. Aquest sistema, de funcionament similar a un SCADA, es dissenyarà mitjançant programació web amb eines que implementin protocols estàndard de tal manera que es puguin enviar consignes i llegir les variables necessàries. Inicialment, per tal de no actuar directament sobre la maqueta, es planteja virtualitzar el sistema fent un model del motor i així poder assegurar el funcionament del sistema de comunicacions de manera prèvia.

Abast:

La part principal del projecte es tractarà del disseny i implementació d'un software que ens permeti llegir i controlar les variables d'entorn que té el motor. Per poder portar a terme el projecte s'hauran de tenir en compte els següents conceptes:

- Estudi i disseny de l'estructura genèrica del sistema.
- Estudi del llenguatges de programació necessaris per al disseny de la pàgina web i del llenguatge que s'utilitzarà per al servidor que ens comunicarà amb el motor.
- Estudi i disseny per la modelització del motor DC.
- Disseny i implementació d'una API per la comunicació entre components del sistema. Per la comunicació del motor amb els clients s'utilitzarà la lògica de 1 servidor per *n* motors, on el servidor gestionarà el conjunt i mirarà la disponibilitat de cadascun.
- Estudi i disseny del hardware necessari per la posada en marxa amb la maqueta del motor.
- Virtualització i simulació del motor per dotar de robustesa el sistema.

Justificació:

La principal motivació per realitzar el projecte es la de donar comoditat a l'usuari que vulgui comunicar-se amb les maquetes dels motors. En aquests moments, es fa servir Matlab com a programa i interfície per al seu control i monitorització i per això, crec que és una molt bona oportunitat per implementar una pàgina web que et faciliti aquest servei.

Pel que fa a la part més tècnica, aquest projecte crec que acapara un ventall molt gran de llenguatges de programació i això és una aspecte que em genera molt d'interès. Per una banda, considero important tenir nocions bàsiques tan de *Front-End* com de *Back-End* i crec que aquest projecte em permetrà aconseguir-ho. D'altra banda, serà necessari utilitzar coneixements adquirits durant el grau pel que fa al control tant del model com de la maqueta.

2 Estat de l'art

2.1 Descripció general

La idea de fer aquest projecte es deu principalment a una pràctica de laboratori que es va fer durant el grau. En termes generals, es tractava de controlar un motor elèctric a través de Matlab (Simulink). L'alumne disposava d'una tarja d'adquisició que permetia la comunicació entre programa i motor i, d'aquesta manera, es podia enviar variables actuant sobre el motor i rebre les magnituds desitjades per poder tractar-les.

Aquesta estructura, avui en dia, té un gran potencial. En l'àmbit industrial hi ha moltes empreses emergents que es dediquen exclusivament a oferir aquest servei de control, monitorització i tractament de dades. Els clients que poden necessitar aquest producte poden ser de sectors molt diferents; fàbriques industrials, hospitals, plantes elèctriques, etc. L'aspecte que tots tenen en comú es que busquen la excel·lència en la **optimització** del seu procés de producció.

2.2 Sistemes SCADA

La necessitat que hi ha en el món industrial de optimitzar els processos de producció no és cap novetat, ja que és un dels factors principals que afecten als beneficis de l'empresa. Per això, els sistemes de control i supervisió coneguts como SCADA sempre han estat present.

Els sistemes SCADA (Supervisió, Control i Adquisició de Dades) tal i com el seu nom indica són utilitzats per controlar i supervisar processos industrials a distància. L'estructura que solen tenir és una combinació de software i hardware (cursosaula21, 2022).

En primer lloc es porta a terme la adquisició de dades de la planta. Per fer-ho s'acostuma a utilitzar un PLC que, rebent totes les senyals dels sensors i actuadors, fa de controlador lògic de la planta. Per la intercomunicació amb el PLC es solen posar RTUs (Unitats Terminals Remotes).

Des de que es fan servir els SCADA, la estructura dels components ha anat evolucionant de manera que es puguin oferir més solucions al client. La part que més ha canviat i que és més fàcil de veure la diferència és la interfície on l'usuari monitoritza la seva planta. En els inicis solament es feia servir un HMI (Human-Machine Interface) dintre de la planta i des d'allí, l'operari s'encarregava de supervisar-ho. Si la planta era massa gran es descentralitzaven les dades i es supervisaven les parts de la planta per separat (cada operari controlava la seva línia de producció).

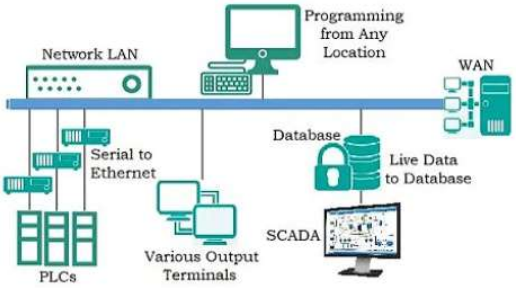

Figura 1. Estructura SCADA actual

En canvi, tal i com es pot veure a la Figura 1 (Raghvendra, 2022) avui en dia s'acostuma a optar per la monitorització mitjançant una aplicació o una pàgina web. D'aquesta manera, es pot controlar, configurar o simplement estar al corrent de que esta passant sense haver de estar en la planta. Aquest canvi ha implicat que es pugui incorporar una base de dades entre l'adquisició de dades i l'HMI. Això permetrà tenir guardat tot l'historial desitjat i poder tractar les dades de la manera que es cregui pertinent; optimitzar el procés, comparar resultats, detectar anomalies, etc.

2.3 Entorn de treball

El laboratori on es portarà a terme el projecte està situat a la primera planta de l'edifici TR11 del campus ESEIAAT. És un laboratori per assignatures de control i automàtica on es fan pràctiques per aprendre a controlar actuadors. L'alumnat pot escollir si vol treballar sobre dos tipus d'actuadors diferents, els dipòsits d'aigua o els motors DC, que són els que s'utilitzaran en aquesta pràctica. Per al cas dels dipòsits, l'objectiu principal és el control del cabal d'aigua, en canvi, la magnitud a controlar en els motors és la velocitat angular.

Un cop escollit 1 dels 10 llocs de treball per fer la pràctica, l'usuari es troba amb els components necessaris per dur a terme el control; el PC, l'actuador, la tarja d'adquisicions i un bastidor amb diferents mòduls.

- PC: S'utilitza principalment per al tractament de les lectures fetes a l'actuador.
- Actuador: Element que es vol controlar. Serà un dipòsit d'aigua o un motor DC.
- Tarja d'adquisicions: Component que s'encarrega de fer les lectures i enviar les consignes necessàries. També fa de convertidor AD/DA.
- Bastidor: Té diferents mòduls que faciliten el control sobre l'actuador.

Figura 2. Lloc de treball

2.4 Hardware

2.4.1 PCI-1711

Per l'adquisició de dades es fa servir la targeta **Advantech PCI-1711**. La principal funció que te la placa és la de comunicar el motor amb el PC mitjançant el bus PCI. Les característiques principals i que són d'interès per fer el projecte són les següents:

- 16 entrades analògiques.
- 2 sortides analògiques.
- 16 entrades digitals.
- 16 sortides digitals.
- 12 bits de resolució
- Freqüència de mostreig màxima 100KHz.
- Llenguatges de programació compatibles: C#,C++, LabVIEW, VB.Net, BCB, MFC, VB6, Delphi, Java, Matlab, Qt.

Les entrades i sortides serviran per poder llegir les magnituds del motor. Com més canals de lectura tingui, més lectures diferents es podran fer. La tarja farà una lectura analògica que la convertirà a digital per a que pugui ser processada pel PC.

Aquesta funció la farà el convertidor AD/DA que porta incorporat. Els bits de resolució i la freqüència màxima de mostreig indiquen la precisió amb la que es podrà dur a terme aquesta conversió. El fet que el convertidor sigui bidireccional (que pot convertir el senyal digital a analògic i viceversa) és molt important. Si l'objectiu del projecte fos solament la supervisió/monitorització de la maqueta, amb el digital-analògic seria

suficient. En canvi, com també es voldrà actuar sobre el motor per tal de controlar-lo, serà necessari passar el senyal de digital a analògic.

Pel que fa al llenguatge de programació hi ha una gran varietat que són compatibles per poder escollir. N'hi ha de molt versàtils com són Java, C# o C++, que serien idonis pel que fa a la comunicació amb el servidor ja que podrien tenir el mateix llenguatge. Però també hi ha les opcions de LabView o Matlab que tenen una aplicació més concreta, un entorn d'usuari més senzill i per tant, poden ser més pràctics per utilitzar models de simulació.

2.4.2 Motor DC

És un motor de la marca Alecop Group que està pensat per utilitzar-lo en un entorn didàctic. Està dissenyat de manera que facilita a l'usuari estudiar el seu procés de control; tant el de posició com el de velocitat.

Tal i com es veu a la figura 2.4.2, la maqueta es pot dividir en 4 parts:

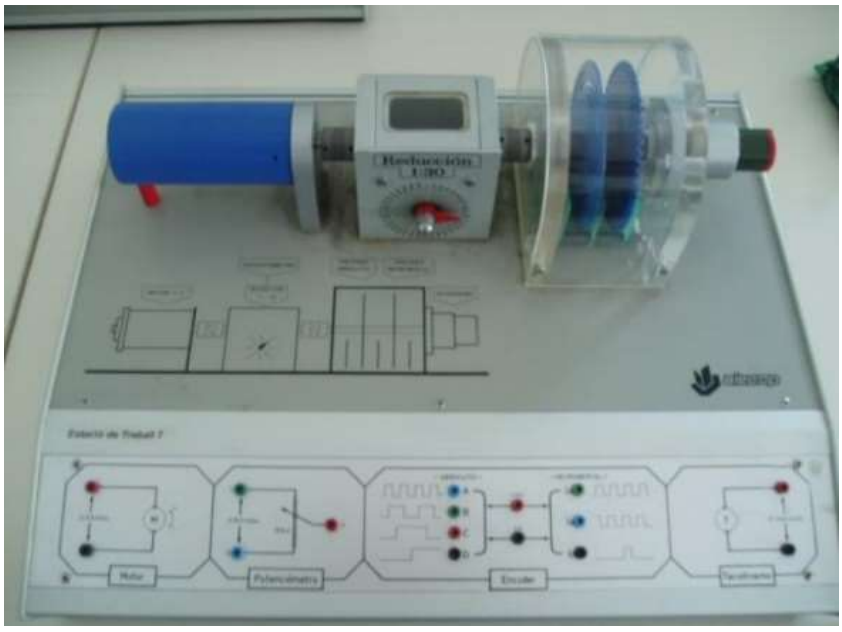

Figura 3. Motor DC

- El motor de corrent continu que se li haurà de proporcionar la potència requerida mitjançant el *driver*.
- Un reductor, per engranatges, de raó 1/30 que s'utilitzarà juntament amb el sensor de posició per efectuar el control de posició angular del motor.
- *Encoders*: Per la realització de les lectures. En el cas de la posició s'utilitzarà l'*encoder* absolut i, per la velocitat, l'incremental.

- *Tacodinamo*: Dona una tensió que serà proporcional a la velocitat de gir del motor i que servirà pel control de velocitat

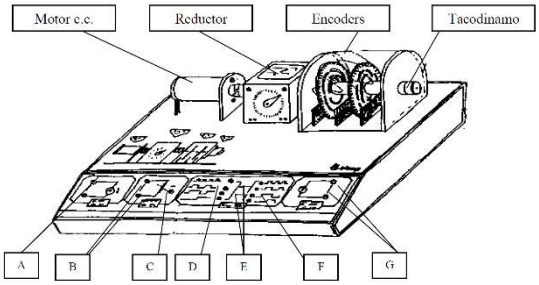

Figura 4. Parts del motor DC

Les entrades/sortides tenen la següent finalitat:

- A) Entrada d'alimentació del motor DC.
- B) Entrada d'alimentació del sensor de posició. El rang de tensions ha de coincidir amb el de la consigna de posició.
- C) Lectura del sensor de posició. Sortida que dona un voltatge proporcional a la posició angular del motor.
- D) Lectura de l'encoder absolut: Sortida que dona la posició angular de manera directa.
- E) Alimentació de l'encoder: Entrada d'alimentació a l'encoder, entre 0 i 5 Volts.
- F) Lectura de l'encoder incremental: Sortida que s'utilitza per conèixer la velocitat del motor mitjançant increments.
- G) Tacodinamo: Sortida de tensió proporcional a la velocitat de gir, entre 0 i 10 Volts.

2.4.3 Bastidor

El bastidor consta de diferents mòduls que permeten a l'usuari controlar amb més comoditat el motor.

- Voltímetre: És un voltímetre digital que et facilitat les lectures desitjades. Té 4 canals de lectura que pots escollir mitjançant un selector rotatiu.
- Consigna: Aquest mòdul és l'encarregat d'enviar una consigna al motor. Els senyals es poden generar en forma de graó i de rampa. L'usuari pot seleccionar la magnitud i el pendent de la rampa. Un cop ja s'ha configurat la consigna, el senyal ha de passar pel driver ja que ha de tenir la potència necessària per alimentar l'actuador.

- Corrector: Aquest mòdul permet el control PID. Té diferents potenciòmetres amb els que l'usuari ajusta els valors de la K_p, K_i/s i $K_d \cdot s$. Un cop es tenen configurats, es fa la combinació desitjada; P, PI o PID. Aquest bloc no es farà servir en la solució proposada. L'ajust de les variables de control (K_p, K_i, i K_d) es farà digitalment.
- Generador: En cas que sigui necessari generar consignes que requereixen anàlisis freqüencials es fa servir aquest mòdul. Es pot adjustar la freqüència, la amplitud i el tipus de senyal; triangular, sinusoidal o quadrada.

Figura 5. Bastidor

2.5 Software

Per millorar el control i supervisió que actualment es fa mitjançant Matlab s'ha de dissenyar un sistema que pugui enllaçar el motor amb la interfície web que utilitzarà l'usuari. Per aconseguir-ho, s'ha de pensar i decidir quins llenguatges de programació s'utilitzaran. En aquest apartat, s'exposaran les opcions que es contemplen i el que ens pot aportar cadascuna.

Per contextualitzar els llenguatges de programació que es faran servir s'han de separar segons si formen part del *front-end* o del *back-end*. Aquest dos termes són molt utilitzats en el món del desenvolupament web ja que són imprescindibles per al bon funcionament d'una web. Des del punt de vista teòric es poden explicar per separat però es necessari que treballin conjuntament.

2.5.1 Front - End

En termes generals, es pot dir que el *front-end* es tot allò que l'usuari pot veure en la web. El text, els colors, la interacció, el disseny, etc. El principals objectius d'aquesta part de la web es aconseguir una bona funcionalitat i que el disseny sigui amigable per l'usuari (user-friendly) (geeksforgeeks, 2022). Per aconseguir aquestes funcionalitats es fan servir els següents llenguatges de programació: HTML, CSS i JavaScript

- *HTML (HyperText Markup Language)*

HTML és el llenguatge que dona estructura i significat a la web. Mitjançant un llenguatge de marques es va definint com es vol elaborar la web de manera que, combinant dades i etiquetes es pot aconseguir la estructura desitjada (developer.mozilla, 2022).

<title>Títol del document</title>

<head>Capçalera del document</head>

Et permet tractar la web com si fos un document; capçalera, cos, paràgrafs, títols, imatges, etc. En definitiva, és el component més bàsic de la web i amb el que tothom hauria de començar si vol dissenyar una web. El terme de hipertext fa referència als enllaços que connecten pàgines web entre sí.

- *CSS (Cascading Style Sheets)*

Un cop es té el document estructurat mitjançant HTML, s'utilitza CSS per descriure l'estil del contingut que es vol mostrar per pantalla. Aquest llenguatge de disseny web ofereix al desenvolupador web la possibilitat de augmentar la seva llibertat a l'hora de donar estil a la pàgina web. Facilita els canvis bàsics com són els de color, font i mida o el fet de donat més versatilitat per crear columnes o espais estratègics dintre del document.

Per tal d'exemplificar el fet de que CSS va de la mà de HTML, a continuació, es mostrarà un petit cas on el document(HTML) es veu afectat per un canvi d'estil produït per CSS.

Taula 1. Exemple d'ús de CSS amb HTML

Codi:	Explicació:
Paràgraf	Codi original HTML: es mostrarà amb les propietats original ja que no s'ha fet cap canvi.
p{color: white; font-family: verdana; font-size: 20px;}	Mitjançant CSS es canvia el color, la font i la mida de la etiqueta *p.* A partir d'ara, totes les etiquetes que surten en HTML tindran aquest nou format de manera automatitzada.

- *JavaScript*

Un cop explicats els llenguatges HTML i CSS, la pàgina encara és un lloc web estàtic on no hi ha dinamisme. Què passa quan prems un botó? I si vols ingressar un valor? Tots aquest factors que són conseqüència de la interacció amb l'usuari, es solucionen amb JavaScript.

Es diu que es un llenguatge de programació interpretat ja que no és necessari tenir una consola on compilar constantment el codi, amb qualsevol navegador es poden fer les comprovacions oportunes.

Amb aquest llenguatge de programació l'usuari podrà fer modificacions sobre el DOM (document object model) de diferents maneres:

- JS en línia: En els documents HTML hi ha la possibilitat de incloure esdeveniments dintre de les etiquetes. Aquests atributs s'acostumen a programar en JS. En l'exemple que surt a continuació, l'esdeveniment "onclick" farà que el botó produeixi una alerta.

```
<button onclick="alert('Segur que vols continuar? ')">D'acord</button>
```

- JS intern <script>: En cas de que no es vulgui incloure el codi dintre del element, es pot afegir a part en una etiqueta anomenada <script>.

```
<script>
function(){alert("I am inside a script tag")}
</script>
```

- JS extern: Hi ha casos on pot interessar separar el codi en JS en arxius separats. Això permet gestionar de manera més ordenada el codi i mantenir una certa estructura que el farà més entenedor. L'atribut "src" de l'etiqueta <script> és el que permet adjuntar l'arxiu JS en el document HTML.

```
<script src="./script.js"></script>
```

Depenent de quin tipus de web es vulgui crear, aquests 3 llenguatges es podrien quedar curts. Normalment, es necessita un *framework* que ajudarà amb el disseny i l'estructura de la pàgina. Aquests sistemes permeten al programador reutilitzar una estructura genèrica predefinida i adaptar-la al seu projecte. Els principals *frameworks* o llibreries que s'utilitzen actualment són els següents (Degiovannini, 2005):

- React
- Angular
- Vue.js
- jQuery

Tots 4 són molt similars ja que s'executen amb Java Script i ofereixen una funcionalitat molt semblant. A continuació, s'exposaran els punts més destacables de cadascun.

React:

ReactJS, tot i ser una biblioteca, és considerat un dels millors *frameworks* en l'actualitat ja que consta d'una arquitectura amb components excel·lents. Tot i ser un software lliure, els creadors han estat els desenvolupadors de Facebook. La principal particularitat d'aquesta interfície és que tota la aplicació s'executa en una mateixa pàgina web. Això dona a l'usuari una experiència més fluida, ja que no s'ha de moure per diferents enllaços; com si fos una aplicació mòbil o d'escriptori (reactjs, 2022). Actualment, les empreses més punteres en el sector tecnològic(Facebook, Apple, Instagram, Netflix, etc.) el fan servir.

Angular:

En aquest cas, AngularJs és un *framework* que va ser creat per Google. L'utilitat i estructura és molt pareguda a React, ja que també es fa servir per tindre tota la interfície en una sola pàgina web (angular, 2022). La principal diferència radica en que al cap i a la fi, React solament es una bibilioteca i Angular és un *framework* complet. Això fa que Angular possiblement tingui un grau superior de complexitat.

Vue:

VueJs és el més recent dels 4. Tot i tenir la interfície en una sola pàgina com els anteriors, té una característica que el fa destacar sobre la resta. El fet de que sigui un framework progressiu, permet al desenvolupador passar i adaptar el seu projecte a Vue. El creador és un desenvolupador independent anomenat Evan You (vuejs, 2022).

jQuery:

JQuery és una excepció ja que no és un framework com a tal. És una biblioteca que va aparèixer al 2006 per facilitar la gestió de les webs amb una idea diferent al que poden ser Vue, Angular i React. L'objeciu principal és facilitar al programador la manera d'interactuar amb els documents HTML, gestió d'esdeveniments, etc. Amb Vanilla JavaScript (ús de JS sense cap tipus de llibreria) la programació es feia molt pesada ja que cada cop es requerien pàgines web més complexes. Avui en dia jQuery s'ha tornat la biblioteca més utilitzada en JS ja que és primordial aconseguir certes funcionalitats.

2.5.2 Back - End

Quant l'usuari vol utilitzar una funcionalitat que veu mitjançant el *front-end* hi ha una lògica al darrere que haurà de actuar per a que la web funcioni correctament. Aquesta lògica, és el que s'anomena *back-end*. Com el seu propi nom indica és tot allò que no es veu, però que està funcionant constantment; comunicació amb el servidor,

comunicació amb la base de dades, accions lògiques, gestió de la seguretat de la web, etc (Presta, 2022). Els 3 llenguatges principals que s'utilitzen avui en dia per cobrir aquesta part de la web són: JavaScript(Node.js), PHP y C#.

Node.js:

El fet de tenir una opció per seguir utilitzant la mateixa sintaxis (JS) al *back-end*, és una de les principals raons que han fet que, en l'actualitat, Node.js sigui la primera opció per a la gran majoria de programadors. Això fa que hi hagi una gran comunitat i que, per tant, sigui més fàcil trobar informació i resoldre possibles *bugs*. És un llenguatge que tot i ser relativament nou, ha estat molt ben acollit per la comunitat de desenvolupadors, sobretot els que ja utilitzaven JS en el *front-end*.

PHP:

PHP és ell llenguatge per excel·lència quan es parla de programació de servidor. Es va crear al 1994 i des de llavors ha estat la referència per als llenguatges *back-end*. Avui en dia el 80% de les webs fan servir PHP. El fet de que sigui tan versàtil fa que totes les persones que s'interessen per la programació web i que venen d'altres branques, l'escullin com a primera opció. Pel que fa a la seguretat també és dels millors.

C#:

C- Sharp és també un llenguatge molt present en la programació *back-end*. La seva principal virtut és la excel·lent compatibilitat amb els servidors de Windows. Això es deu a que ha estat creat per els desenvolupadors de Microsoft. La sintaxis és senzilla d'aprendre ja que deriva de C i C++, llenguatges amb els quals els programadors solen estar acostumats. Pel que fa a les seves limitacions, es pot destacar el fet de que et limita a utilitzar una computadors Windows i amb la macro .NET. Avui en dia, aquesta manca de flexibilitat per al programador limita molt el seu ús.

3 Metodologia

En aquest apartat s'exposaran els diferents blocs que es tocaran durant el transcurs del projecte. En cada àmbit s'escollirà el llenguatge que més s'adapti a les necessitats del treball de manera que es puguin complir els objectius establerts.

Pel que fa al *front-end*, HTML, CSS i JS, no s'hauran de comparar amb cap altre llenguatge ja que són primordials per al disseny i programació de qualsevol pàgina web. El que sí que és comparable és la manera en la que es vol programar la dinàmica de la web.

3.1 Comparació de frameworks front-end

Com s'ha comentat en l'apartat anterior, l'ús de JavaScript es pot implementar amb diferents frameworks depenent del tipus de web que es vulgui crear. Per tant, en aquest cas, quina seria la millor opció? Per decidir quin llenguatge utilitzar, es compararan totes les opcions seguint uns criteris amb diferent importància. Cada criteri s'avaluarà de 0 a "n" i es multiplicarà per el seu percentatge d'importància. Al final, el llenguatge amb la nota més alta serà el que s'utilitzarà.

Taula 2. Comparació frameworks front-end

Criteris (% d'importància)	**REACT**	**Angular**	**Vue**	**JQuery**
Corba d'aprenentatge (50%)	3	1	2	4
Flexibilitat o escalat (10%)	3	2	4	1
Posada en Marxa (20%)	3	1	2	4
Comunitat/Documentació (20%)	3	2	1	4
Total:	**3**	**1,3**	**2**	**3.7**

Un cop feta la taula comparativa, es pot observar que la millor opció serà la de jQuery. La web que es vol dissenyar ha de complir una funcionalitat molt concreta com és la de llegir i controlar un motor elèctric. Per tant, no serà necessari utilitzar cap *framework* que ajudi amb la estructura de l'aplicació. , sobretot pel temps que pot comportar la seva corba d'aprenentatge.

3.2 Comparació llenguatges back-end

En el cas del *back-end*, es seguirà la mateixa metodologia per conèixer el llenguatge més idoni per al projecte.

Taula 3. Comparació llenguatges back-end

Criteris (% d'importància)	**Node.js**	**PHP**	**C#**
Corba d'aprenentatge (40%)	2	3	1
Flexibilitat o escalat (20%)	3	2	1
Posada en Marxa (20%)	3	2	1
Comunitat/Documentació (20%)	3	2	1
Seguretat (10%)	2	3	1
Total:	**2.8**	**2.7**	**1.1**

En aquest cas, l'idoni seria fer-ho amb Node.js. Una bona opció també seria utilitzar PHP ja que per coneixements previs, és el que comporta un procés d'aprenentatge més ràpid. En qualsevol cas, utilitzar Node.js + jQuery és una gran combinació i una bona oportunitat per aprendre com utilitzar-los conjuntament.

El següent pas per dissenyar una pàgina web de monitorització de processos, seria analitzar quina base de dades escollir. En aquest cas, no serà necessari perquè no es vol una lectura a temps real, sinó graficar la simulació demanada. En les lectures *in-real-time* si que és necessari guardar aquestes dades per analitzar-les posteriorment.

3.3 Comparació llenguatges per l'adquisició de dades

Per l'adquisició de dades, tal i com es pot veure a l'apartat 2.4.1, la targeta d'adquisició permet utilitzar diferents llenguatges de programació. Per tant, tal i com s'ha fet en els casos anteriors, es compararan les millors opcions per saber quina escollir. Actualment, al laboratori, es fa servir Matlab i per tant, això serà un punt important per tenir en compte.

Taula 4. Comparació llenguatges per l'adquisició de dades

Criteris (% d'importància)	**C#**	**Java**	**Matlab**	**LabView**
Corba d'aprenentatge (40%)	2	3	4	1
Posada en Marxa (10%)	3	2	4	1
Comunitat/Documentació (20%)	2	4	3	1
Control de models (20%)	2	1	4	2
Total:	**1.9**	**2.4**	**3.4**	**1.1**

Per tant, la millor opció serà Matlab. El llenguatge és conegut i, a sobre, t'ofereix la possibilitat d'utilitzar Simulink per simular models i evitar actuar sobre el motor directament.

4 Implementació de l'algoritme

En aquest apartat s'explicarà el procés que s'ha seguit per dissenyar i crear el projecte. En comptes de explicar tot el codi de manera continuada, es dividirà l'esquema en petites parts diferenciades. Cadascuna s'haurà programat en un llenguatge diferent i, per tant, tindrà una funcionalitat concreta.

4.1 Esquema general

Tot i que el flux de dades va des de la web fins al motor, és a dir, l'usuari envia una consigna i posteriorment el motor reacciona. Per explicar el procés, es començarà pel final. S'analitzaran totes les comunicacions que hi ha, començant per la del motor amb Matlab i acabant amb la del *front-end* de la web.

4.2 Motor DC – Matlab (Simulink)

Un cop Matlab rep la petició de Python de que hi ha una ordre per accionar el motor s'executen una sèrie d'arxius depenent de si es vol treballar en llaç obert o en llaç tancat. En tots dos casos, el flux d'informació passa pel mateix tipus d'arxius:

Init. (.m) -> Intermediari (.m) -> Simulador (.slx)

El *iniciador* s'encarrega de iniciar a 0 totes les variables a les que Python necessitarà accedir. Al posar-les a zero, estaran disponibles al *Workspace* de Matlab i per tant, podran ser assignades pel valor desitjat.

El *intermediari* farà executar l'arxiu Simulink i guardarà la lectura en la variable a la qual accedeix Python, així es podrà enviar a la gràfica de la web.

El simulador és un arxiu Simulink que permet enllaçar el motor amb el software mitjançant la targeta d'adquisició. També et possibilita l'opció de posar un controlador PI en el cas del llaç tancat.

A continuació s'explicarà el procés que segueix en el cas del llaç tancat amb el codi corresponent:

1) Inicialitza les variables que es necessitaran.

initcl.m

```
1  outputReadCl = 0;
2  simTimeReadCl = 0;
3  SetpointCl = 0;
4  InvSetpointCl = 0;
5  Kp = 0;
6  Ki = 0;
```

Figura 6. Codi iniciador.m

2) Executa l'arxiu Simulink i guarda la lectura en la variable outputReadCl.

simReadCl.m

```
1   out = sim('readValuesCl.slx');
2   outputReadCl = out.outputReadCl;
```

Figura 7. Codi intermediari.m

3) Es simula el model de Simulink.

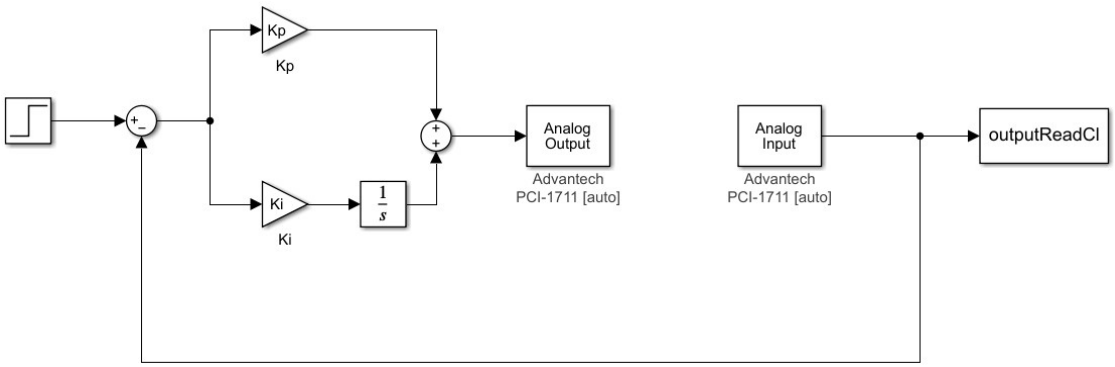

Figura 8. Model Simulink llaç tancat

El graó d'entrada tindrà el valor de la variable Setpoint. El controlador agafarà els valors de les constants Kp i Ki. Un cop la senyal passa per el controlador va al *Analog Output* de la targeta d'adquisició per poder actuar sobre el motor. Durant la simulació s'agafa la lectura mitjançant l'*Analog Input.*

Un cop acabada la simulació Python agafa la variable *outputReadCl* i l'envia a la web.

4.3 Matlab (Simulink) – Python

Per poder connectar aquest dos programes s'ha de buscar un paquet o extensió que permeti fer d'enllaç. Per aquesta funcionalitat, Matlab té una API anomenada MATLAB Engine (mathworks, 2022). A grans trets, es podria dir que permet al programador fer servir la consola de comandes/motor de càlcul de Matlab en un programa extern; en aquest cas Python.

4.3.1 Posada en marxa

Per la posada en marxa s'ha d'instal·lar tant Matlab com Python. Depenent de quina versió es té de Python, s'ha de buscar un a versió compatible de Matlab i viceversa. MathWorks ofereix una taula amb totes les combinacions que es poden fer per aconseguir que MatlabEngine funcioni correctament. Per aquest projecte, es farà servir la primera combinació(R2022a – Python 3.9).

MATLAB Version	Compatible Versions of Python 2	Compatible Versions of Python 3
R2022a	2.7	3.8, 3.9
R2021b	2.7	3.7, 3.8, 3.9
R2021a	2.7	3.7, 3.8
R2020b	2.7	3.6, 3.7, 3.8
R2020a	2.7	3.6, 3.7
R2019b	2.7	3.6, 3.7
R2019a	2.7	3.5, 3.6, 3.7
R2018b	2.7	3.5, 3.6
R2018a	2.7	3.5, 3.6
R2017b	2.7	3.4, 3.5, 3.6
R2017a	2.7	3.4, 3.5
R2016b	2.7	3.3, 3.4, 3.5
R2016a	2.7	3.3, 3.4
R2015a	2.7	3.3, 3.4
R2015b	2.7	3.3, 3.4
R2014b	2.7	3.3

Figura 9. Compatibilitat versions (Python - Matlab)

Un cop s'han instal·lat els dos programes i s'ha aconseguit la compatibilitat necessària, es procedeix a instal·lar l'API desitjada. Per fer-ho, s'ha d'accedir al terminal de Windows i seguir els següents passos:

- `cd "matlabroot\extern\engines\python"`
- `python setup.py install`

4.3.2 Comunicació entre els programes

Com s'ha mencionat anteriorment, el flux de dades va de la web fins al motor. Per tant, la comunicació s'establirà des de Python. A continuació, es mostrarà com fer-ho:

1) Importa el mòdul Matlab Engine prèviament instal·lat.

- `import matlab.engine`

2) Inicialitza la consola de Matlab per poder actuar de manera externa i assigna la inicialització a una variable.

- `matlabEngine = matlab.engine.start_matlab()`

Mitjançant "matlabEngine" ja es podrà fer qualsevol operació o comandament acceptat per Matlab.

3) Per comprovar-ho, es calcularà l'arrel quadrada de 49.

- `matlabEngine.sqrt(49)`

El que la consola mostra per pantalla és el següent:

```
>>> matlabEngine.sqrt(49)
Check for incorrect argument data type or missing argument in call to function 'sqrt'.

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\Usuario\AppData\Local\Programs\Python\Python39\lib\site-packages\matlab\engine\matlabengine.py", line 7
0, in __call__
    return FutureResult(self._engine(), future, nargs, _stdout,
  File "C:\Users\Usuario\AppData\Local\Programs\Python\Python39\lib\site-packages\matlab\engine\futureresult.py", line 6
7, in result
    return self.__future.result(timeout)
  File "C:\Users\Usuario\AppData\Local\Programs\Python\Python39\lib\site-packages\matlab\engine\fevalfuture.py", line 82
, in result
    self._result = pythonengine.getFEvalResult(self._future,self._nargout, None, out=self._out, err=self._err)
matlab.engine.MatlabExecutionError: Undefined function 'sqrt' for input arguments of type 'int64'.
```

Figura 10. Error conversió tipus de dades

Tal i com es veu a la figura, Python no aconsegueix fer l'operació correctament. El motiu, apareix en l'última línia.

matlab.engine.MatlabExecutionError: Undefined function 'sqrt' for input arguments of type 'int64'.

El motiu de l'error és per la conversió de dades entre Python y Matlab. És a dir, el tipus de data que enviem no es manté quan Matlab el rep. En aquest cas, s'envia un *int* però el que rep es un *int64* i, per aquest motiu, no aconsegueix fer l'arrel quadrada. MathWorks ofereix una taula de totes les conversions possibles, que ajudarà a saber com s'han d'enviar les dades.

Taula 5. Conversions de dades (Python - Matlab)

Entrada Python	**Resultat Matlab**
float	double
complex	complex double
int	int64
float(NaN)	NaN
float(inf)	Inf
bool	logical
str	char
dict	char

Un cop sabuda aquest informació, es tornarà a fer l'operació. Com es vol que Matlab rebi un *int* o *double*, des de Python, s'ha d'enviar un *float*.

```
>>> matlabEngine.sqrt(49.0)
7.0
```

Figura 11. Solució correcta

Tal i com mostra la figura, el càlcul s'ha fet perfectament. Per tant, ja es pot dir que s'ha aconseguit la comunicació entre els dos programes.

4.3.3 Codi per enviar consigna – adquirir dades

Abans de començar amb en el codi, cal aclarir que al projecte hi haurà dos arxius .py. El primer (connectMatlab.py) per treballar en open loop, i un altre per treballar en closed loop (clconnectMatlab.py). L'estructura del codi és igual, la diferència es troba en la quantitat de variables que assignem. En open loop solament enviem la consigna, en canvi, en closed loop també enviem el valor de Kp i Ki. Per tant, cadascun es comunicarà amb el seu model de Simulink corresponent; *simRead* o *simReadcl*. Per explicar el codi, s'utilitzarà el del arxiu clconnectMatlab.py:

1) Importa el paquet MatlabEngine que permetrà connectar-se amb Matlab (Simulink).

```
- import matlab.engine
```

2) Importa el mòdul *sys* que permetrà connectar-se amb Node.Js

```
- import sys
```

3) Rep les consignes enviades des de la web

```
- arrayPy = sys.argv[1].split(',')
```

La variable *argv* del mòdul *sys* retorna una llista amb tots el arguments que s'han passat. En aquest cas, es rep un *string* amb totes les variables. Per classificar-les dintre de l'arxiu, es fa un split(',') aprofitant que estant separades per comes. D'aquesta manera, es tindrà una matriu amb una variable en cada posició. Aquesta línia s'analitzarà millor posteriorment, ja que també forma part d ela connexió entre Python – NodeJs.

4) Inicialitza uns instància de Matlab en una variable

```
- matlabEngine = matlab.engine.start_matlab()
```

5) Executem el arxiu de matlab initcl.m

```
- matlabEngine.initcl(nargout=0)
```

Amb nargout=0 s'indica que no s'espera cap argument de retorn. El arxiu simplement és un iniciador de totes les variables que es necessitaran. D'aquesta manera, el *workspace* al que s'accedirà posteriorment ja estarà disponible.

6) Accedeix a les variables del workspace i s'assignen els valors marcats per la web.

- `matlabEngine.workspace['SetpointCl'] = float(arrayPy[0])`
- `matlabEngine.workspace['InvSetpointCl'] = -float(arrayPy[0])`
- `matlabEngine.workspace['Kp'] = float(arrayPy[1])`
- `matlabEngine.workspace['Ki'] = float(arrayPy[2])`

Abans de assignar el valor, s'ha de convertir en *float* per el motiu explicat anteriorment. Cada variable està en una posició diferent de la matriu arrayPy. Aquesta matriu és la creada en el punt 3.

7) Un cop totes les variables del model de Simulink estan inicialitzades, s'executa l'arxiu simReadCl.slx.

- `matlabEngine.simReadCl(nargout=0)`

En aquest cas, com tampoc s'espera cap argument de sortida; nargout = 0.

8)Retorna la lectura de la simulació completa sobre el motor.

- `print(matlabEngine.workspace['outputReadCl'])`

Aprofitant el mòdul sys utilitzat al apartat 3 per rebre la petició de la web s'utilitza la funció print() per retornar la lectura emmagatzemada en la variable outputReadCl creada pel model.

4.3.4 Codis complets

connectMatlab.py > ...

```python
import sys                                          #node.js connectivity
import matlab.engine                                #matlab connectivity
setpointPy = float(sys.argv[1])                     #we get the simulation/setpoint time from the server
matlabEngine = matlab.engine.start_matlab()         #start matlab engine
matlabEngine.init(nargout=0)

matlabEngine.workspace['Setpoint'] = setpointPy       #important to send as float!!!!
matlabEngine.workspace['InvSetpoint'] = -setpointPy

matlabEngine.simRead(nargout=0)
plotArrayRead = matlabEngine.workspace['outputRead']


print(matlabEngine.workspace['outputRead'])
```

Figura 12. Codi Python Open Loop

```python
clconnectMatlab.py > ...
import sys                                          #node.js connectivity
import matlab.engine                                #matlab connectivity

arrayPy = sys.argv[1].split(',')

matlabEngine = matlab.engine.start_matlab()         #start matlab engine
matlabEngine.initcl(nargout=0)

matlabEngine.workspace['SetpointCl'] = float(arrayPy[0])       #important to send as float!
matlabEngine.workspace['InvSetpointCl'] = -float(arrayPy[0])
matlabEngine.workspace['Kp'] = float(arrayPy[1])
matlabEngine.workspace['Ki'] = float(arrayPy[2])

matlabEngine.simReadCl(nargout=0)

print(matlabEngine.workspace['outputReadCl'])
```

Figura 13. Codi Python Closed Loop

4.4 Python – Node.js

La comunicació entre aquests dos programes/llenguatges fa de bus comunicatiu entre el que veu l'usuari a la web i el que passa al model. Node.js podria fer de *back-end* en solitari si no fos necessari passar per Matlab. O per exemple, si Matlab tingués una extensió o paquet que comuniqués directament amb node.js.

A part de la funció de transferència de dades, Node.js fa de servidor web per poder allotjar-la en un port en concret. Aquesta funció s'explicarà més detalladament en l'apartat 4.5.

4.4.1 Posada en marxa

Primer de tot, s'han de tenir instal·lats tots dos programes. En el cas de Python, ja s'ha fet en l'apartat anterior. Node.js es pot descarregar i instal·lar directament des de la pàgina oficial; https://nodejs.org/es/

4.4.2 Comunicació

Per establir comunicació entre aquests dos programes s'ha de definir quin és el sol·licitant i quin es el receptor; també es podria contemplar l'opció de cada programa fes totes dues funcions. L'objectiu del projecte es fer funcionar el motor en unes condicions concretes i, posteriorment, adquirir les dades llegides. Per tant, això es podrà simplificar de la següent manera:

Figura 14. Comunicació back-end

Tal i com s'indica a la figura, Node.js s'encarregarà d'enviar la petició quan la web li ho indiqui. La sol·licitud enviada inclourà la informació necessària per a que Python pugui fer funcionar el motor en les condicions desitjades. Per poder retornar la informació, en comptes d'iniciar una altra petició de manera que Python sigui el sol·licitant, el que es farà serà retornar-ho mitjançant la mateixa creada per Node.js. D'aquesta manera, es conserva la relació sol·licitant – receptor que s'havia definit. En el següent apartat s'explicarà amb el codi per poder-ho exemplificar de millor manera.

4.4.3 Codi per establir comunicació

El codi fet amb JS està en l'arxiu server.js. Depenent de si la sol·licitud es vol fer per accionar el motor en llaç obert o en llaç tancat, s'utilitzarà una petició que crida a l'arxiu *connectMatlab.py* o al *clconnectMatlab.py.* Com a l'apartat 4.3.3 s'ha explicat el cas del llaç tancat, es continuarà amb el mateix exemple:

1) Importa Express.js a la web (expressjs, 2022).

```
- const express = require('express')
```

Express.js es un *framework* bàsic i senzill que ajuda molt amb a la gestió i estructura de la web (ifgeekthen.nttdata, 2022).

2) Es configura com a variable.

```
- const app = express()
```

3) Amb la funció .get el servidor està atent per a que enviar la sol·licitud quan la web li ho indiqui.

```
- app.get('/receivecl', (req, res) => {                ...              })
```

Quan en la ruta */receivecl* hi hagi una petició per part del *front-end* de la web, la funció *.get* actua. La variable *req* conté les variables que s'hauran d'enviar al motor, en canvi, la variable *res* és la que s'utilitzarà per retornar al *front-end* la informació rebuda de la maqueta. En resum, es diria que *req* és el que el servidor rep i *res* és el que retorna, un cop el motor ha acabat. A continuació es mostrarà el que hi ha dintre de la funció.

4) Desa en la variable "arrayClPy" les variables enviades per la web.

```
- var arrayClPy = req.query.time.split(',');
```

Com les variables arriben en format *String* i separades en comes, es torna fer un *split().*

5) Importem el mòdul *child_process* que permetrà iniciar subprocessos i tractar comandaments Shell (nodejs, 2022). Aquest mòdul serà l'encarregat de connectar amb el arxius Python

- `const spawner = require('child_process').spawn;`

6) Inicia un subprocés per comunicar-se amb l'arxiu de Python.

- `const python_processCl = spawner('python',['./clconnectMatlab.py', arrayClPy]);`

S'aprofita el mètode asíncron *spawn* per iniciar la comunicació amb Python. Aquesta funció requereix de tres variables per a que funcioni correctament:

`child_process.spawn(command[, args][, options]):`

command: El comandament que s'ha d'executar, en aquest cas, obrir la consola de Python.

arg: *String* d'arguments, en aquest cas, l'arxiu que ha d'obrir la consola de Python.

options: Opcions addicionals, en aquest cas, la informació enviada.

7) Un cop Python ha fet accionar el motor, el servidor rebrà la lectura mitjançant la funció *stdout.on*, que ve del *spawner* iniciat al pas 6.

- `python_processCl.stdout.on('data', (dataCl) =>{ ... })`

Per tant, la funció de lectura *stdout.on* serà la encarregada de rebre i desar, en la variable dataCl, la matriu amb tots els valors de lectura del motor. A continuació, es mostrarà el codi que hi ha dintre de la funció.

8) Desem la matriu *dataCl* en la variable *arrayReadCl*.

- `var arrayReadCl = dataCl`

9)Retorna a la web la matriu de lectura aprofitant la mateixa petició que ha enviat al pas 3.

- `res.send(arrayReadCl);`

Tal i com s'indica al pas 3, s'utilitza la variable *res* per retornar la informació sol·licitada. Per fer-ho, la funció *send* s'encarregarà. Aquesta funció es molt versàtil ja que quan no hi ha res a retornar, es pot aprofitar com un indicador que li diu a la web si el procés s'ha fet correctament o no.

4.4.4 Codi comunicació Node.js – Python

```
const express = require('express')
const app = express()

app.get('/receivecl', (req, res) => {

  var arrayClPy = req.query.time.split(',');
  const spawner = require('child_process').spawn;

  console.log('Data sent to python script: ', arrayClPy);

  const python_processCl = spawner('python',['./clconnectMatlab.py', arrayClPy]);
  //console.log(python_processCl);
  python_processCl.stdout.on('data', (dataCl) =>{
      var arrayReadCl = dataCl;
      console.log('Data received from python script closed loop:', arrayReadCl.toString());
      res.send(arrayReadCl);
    // console.log('Data received from python script cleaned:', data);
  });
})
```

Figura 15. Codi comunicació Node.js - Python

4.5 JavaScript – Node.js

En aquest apartat s'explicarà com es comunica el servidor amb la web. Un cop dissenyada la web es necessita un servidor on es pugui executar i fer-la funcionar. Per aconseguir-ho, Node.js es ideal.

4.5.1 Comunicació servidor – front-end

Al ser una web bastant senzilla, la part del servidor solament tindrà dos estats.

1. Obrir la web quan detecti que s'ha introduït l'enllaç adequat.
2. Enviar la consigna un cop sigui executada per l'usuari.

4.5.2 Codi per la primera funcionalitat

Per aconseguir la primera funcionalitat solament actua el servidor, ja que és el que s'encarrega de la execució de la web.

1) Importa express() com s'ha mostrat anteriorment. Aquest mòdul es el que permetrà fer tota la execució d'una manera senzilla i estructurada.

2) Assigna el port en el vols allotjar la web.

- `const port = 3000`

3) Importa el mòdul path(). Permetrà al programador treballar amb directoris. Això és útil perquè cada arxiu dintre de la carpeta de la web te una funció en concret i, per tant, el fet de que la web es pugui redirigir a l'arxiu desitjat és important.

- `const path = require('path')`

4) Carga els arxius *front-end*. D'aquesta manera, quan es produeixi la sol·licitud a la URL, els arxius per a que es vegi correctament la web estaran preparats. Recorda, que la variable app és el mòdul express que s'ha declarat al apartat 4.4.3 (pas 2).

- `app.use(express.static(__dirname + '/public'))`
- `app.use(express.static(__dirname + '/client'))`

En aquest cas, els arxius estan en les carpetes *public* i *client*. En la primera es troben l'arxiu HTML (index.html) per l'estructura de la web i l'arxiu JavaScript (script.js) per donar-li dinàmica a la pàgina. En la segona carpeta està l'arxiu CSS (style.css) que ajuda a millorar l'estètica de la web.

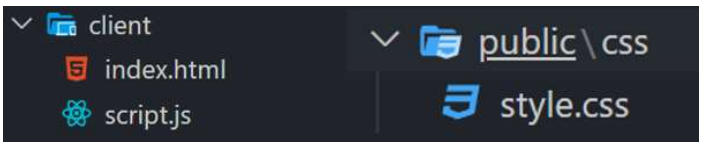

Figura 16. Carpetes front-end

5) Un cop s'entra a l'URL adequada (localhost:3000) la web s'obri amb l'arxiu principal (index.html). Se li sol denominar arxiu arrel ja que és el primer al que accedeix el servidor. A partir d'aquest document, s'executen la resta.

- `app.get('/', (req, res) => {`

  `res.sendFile(path.join(__dirname, '/client/index.html'));})`

L'estructura d'aquesta funció és la mateixa que s'ha explicat en l'apartat 4.4.3 (pas 3).

6) El servidor està atent per quan l'usuari entre al port de la web. Si es fa servir qualsevol altre port que no sigui el 3000, la web no funcionarà.

- `app.listen(port, () => {`
  `console.log(\`Example app listening on port ${port}\`)})`

Un cop fet el codi, el procediment per accedir a la web seria el següent:
Executar la línia: *node server* en la consola -> Obrir l'enllaç *localhost:3000*

4.5.3 Codi per la segona funcionalitat

Quan l'usuari envia la consigna al motor, la web inicia un procés asíncron que s'analitzarà a continuació. Com en els casos anteriors, s'agafarà com a exemple el procés de llaç tancat.

1) L'arxiu *index.html* crida la funció *writeValuesClosedLoop()* de l'arxiu *script.js* en quan es premi el botó de *write*.

```
- <button id="writecl" onclick="writeValuesClosedLoop()"  ..   </button>
- function writeValuesClosedLoop(){              ...              }
```

2) Dintre de la funció writeValuesClosedLoop() s'agafen els valors introduïts per l'usuari. El primer filtre que passen es per assegurar-de que son números i no lletres o alguna valor NaN.

```
- var clSetpoint =  $('#consignacl').val() ? $('#consignacl').val() :
  null;
- var clKp =  $('#kp').val() ? $('#kp').val() : null;
- var clKi =  $('#ki').val() ? $('#ki').val() : null;
```

En les 3 variables es mira si és un valor. En cas que no ho sigui, el contingut de la variable declarada serà *null*.

2) Les variables passen un segon filtre on, si els valors son negatius o null, la web mostrarà un missatge d'error/alerta per pantalla.

```
- if(clSetpoint === null || clSetpoint<0){
        alert("Setpoint not assigned")
  }
- if(clKp === null || clKp<0){
        alert("Kp not assigned")
  }
- if(clKi === null || clKi<0){
        alert("Ki not assigned")
  }
```

3) Si les variables són correctes, es passen al tipus *float*.

```
- clSetpoint = parseFloat(clSetpoint);
- clKp = parseFloat(clKp);
- clKi = parseFloat(clKi);
```

4) Crida a la funció que s'encarregarà de comunicar-se amb el servidor.

```
- requestWriteClosedloopPy(clSetpoint, clKp, clKi);
```

La funció rebrà els tres paràmetres introduïts després de passar el filtre. D'aquesta manera la web s'assegura que, abans de fer la petició, les variables enviades són correctes i per tant, el motor funcionarà correctament.

5) Executa la funció requestWriteClosedloopPy().

```
- function requestWriteClosedloopPy (clSetpointPy, clKpPy, clKiPy) { … }
```

6) Declara la matriu arraycl amb les variables.

```
- var arraycl = [clSetpointPy, clKpPy, clKiPy]
```

7) Creem la URL on voldrem enviar aquesta informació

```
- url = 'http://localhost:3000/receivecl?time=' + arraycl + ''
```

Si es recorda l'apartat 4.4.3, en el pas 3, el servidor ha d'agafar les variables quan detecta que la URL té el nom */receivecl; app.get('/receivecl', (req, res).* Per això, es redirigeix la URL d'aquesta manera. Les variables estaran emmagatzemades al objecte time. Per això, en el pas posterior es feia lo següent; *var arrayClPy = req.query.time.split(',');*

8) Es fa servir un AJAX get() per enviar les variables i, posteriorment, rebre la lectura. Aquesta funció és un mètode que proporciona jQuery i que fa que aquest framework sigui ideal per aquesta aplicació.

```
- $.ajax({                    ...                    })
```

9) Dintre de la funció ajax, s'indica de quin tipus de mètode es tracta (GET o POST) i la url a la qual es vol que la web es dirigeixi.

```
- type:"GET",
- url:url,
```

10) Si la petició s'ha fet correctament, entra en la funció success. Això equivaldria a que el motor ha actuat correctament i que ha retornat els valors de lectura. Aquesta funció ja és l'últim apartat que hi ha dintre del mètode AJAX get().

```
- success:function(datos, status){              ..            }
```

En la funció es rep la variable *datos*, que conté la matriu de lectura, i la variable *status* que indica si la petició s'ha fet correctament; bàsicament s'utilitza com a *feedback* del *back-end.*

11) Convertim a JSON per poder tractar millor la matriu de lectura.

- `datos = JSON.parse(datos)`

12) Declara la matriu on es passaran els valors de lectura. Aquesta matriu serà la que al final es graficarà.

- `var datosCleaned = []`

Aquesta matriu es crea perquè la anomenada datos està en un format que no interessa. El que es vol és una matriu de x posicions on cada posició té el valor de una lectura en un instant determinat. En canvi, el que es rep i el que hi ha a la matriu datos, és una matriu on dintre hi ha x matrius que cadascuna té el seu valor.

13) Es van agafant els valors mitjançant dos bucles i es van introduint en la nova matriu amb la funció push():

```
for (let i = 0; i < datos.length; i++) {
	const dato = datos[i];
	for (let j = 0; j < dato.length; j++) {
		const val = dato[j];
		datosCleaned.push(val)
	}
}
```

14) Un cop es té la matriu amb el format desitjat, s'envia per a que s'executi la gràfica.

- `runChartCl(datosCleaned)`

15) La gràfica es fa mitjançant una API anomenada *Highcharts*. La funció te una estructura predefinida que el programador va omplint segons les característiques que desitgi.

```
let runChartCl = (data = []) => {
    Highcharts.chart('chartcl', {            ...          }
}
```

4.5.4 Codi comunicació servidor – front-end

```
function writeValuesClosedLoop(){

    var clSetpoint =  $('#consignacl').val() ? $('#consignacl').val() : null;
    var clKp =  $('#kp').val() ? $('#kp').val() : null;
    var clKi =  $('#ki').val() ? $('#ki').val() : null;

    if(clSetpoint === null || clSetpoint<0){
        alert("Setpoint not assigned")
    }
    if(clKp === null || clKp<0){
        alert("Kp not assigned")
    }
    if(clKi === null || clKi<0){
        alert("Ki not assigned")
    }
    clSetpoint = parseFloat(clSetpoint);
    clKp = parseFloat(clKp);
    clKi = parseFloat(clKi);
    requestWriteClosedloopPy(clSetpoint, clKp, clKi);
}
```

Figura 17. Codi JavaScript comunicació amb servidor(1)

```
function requestWriteClosedloopPy(clSetpointPy, clKpPy, clKiPy){

    console.log(typeof(clSetpointPy));
    var arraycl = [clSetpointPy, clKpPy, clKiPy]
    console.log(arraycl);
    console.log(typeof(arraycl));
    url = 'http://localhost:3000/receivecl?time=' + arraycl + ''
    $.ajax({
        type:"GET",
        url:url,
        success:function(datos, status){
            console.log({datos})
            datos = JSON.parse(datos)
            var datosCleaned = []
            for (let i = 0; i < datos.length; i++) {
                const dato = datos[i];
                for (let j = 0; j < dato.length; j++) {
                    const val = dato[j];
                    datosCleaned.push(val)
                }
            }
            runChartCl(datosCleaned)
        },
    })
}
```

Figura 18. Codi JavaScript comunicació amb servidor(2)

4.6 HTML – CSS

Per acabar amb el codi, queda explicar la part estàtica de la web. HTML i CSS són els encarregats de donar la estructura i millorar la estètica de la aplicació.

En el cas de HTML s'explicarà el codi per sobre ja que, al ser un llenguatge per etiquetes, és més complicat explicar-ho línia per línia, com s'ha fet amb els altres llenguatges.

4.6.1 Codi HTML

El codi es pot separar en dos grans blocs; capçalera i cos. El <head> s'ha utilitzat per carregar totes les fonts i estils convenients. El <body> és la part on està el codi que dona estructura a la web.

Head:

```
<head><!--save all fonts and styles needed-->
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>TFGWeb</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;700&display=swap" rel="stylesheet">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css">
    <link rel="stylesheet" type="text/css" href="/css/style.css">
</head>
```

Figura 19 Codi capçalera HTML

En aquest apartat es carreguen tant els estils que s'importen de enllaços externs com els que s'han creat pel mateix programador. En el primer cas es troben la majoria de casos, són els que tenen un enllaç en la propietat href. El segon cas es pot trobar en la última etiqueta, l'arxiu /css/style.css s'ha creat des de zero per facilitar el disseny d'algunes parts del codi.

Body:

Per explicar aquesta part, es mostrarà el codi que s'ha fet i la conseqüència de cara a la web. És a dir, d'aquesta manera es veurà d'una manera molt visual que és el que s'aconsegueix amb cada part del codi.

1) Capçalera de la web

```
<body>
    <nav>
        <div class="nav-wrapper">
          <a href="#" class="brand-logo" style="margin-left: 50px">TFG Web</a>
        </div>
    </nav>
```

Figura 20. Títol de la web

2) Selecció de si es vol treballar en llaç obert o en llaç tancat.

```
<section class="openloop-container">
    <div class="row">
        <div class="col s12">
          <ul class="tabs">
            <li class="tab col s6"><a class="active" href="#openloop">OPENLOOP</a></li>
            <li class="tab col s6"><a href="#closedloop">CLOSEDLOOP</a></li>
          </ul>
        </div>
```

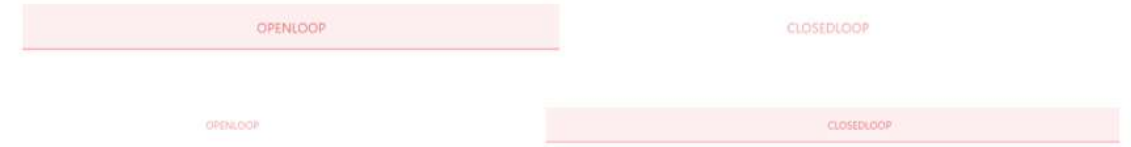

Figura 21. Selecció d'opcions

Depenent de la selecció, es mostrarà la estructura dissenyada per al llaç obert o la del llaç tancat. El canvi de pàgina no implica un canvi de URL. Això dona una sensació de aplicació que avui en dia és molt utilitzada. Aquesta funcionalitat la permet el framework jQuery del que s'ha parlat anteriorment.

3) Estructura llaç obert. Introducció de la consigna i gràfica.

```
<div id="openloop" class="col s12">
    <div class="row">
        <form class="col s12">
          <div class="row">
            <div class="input-field col s6">
              <input placeholder="Setpoint" id="consignaop" type="number" class="validate">
              <label for="consigna">Consigna</label>
            </div>
          </div>
          <div class="row">
            <button id="writeop" onclick="writeValuesOpenLoop()" class="btn waves-effect waves-light" type="button"
              name="action">Write</button>
          </div>
          <div class="row">
              <div id="chart">

              </div>
          </div>
        </form>
      </div>
</div>
```

Figura 22. Estructura llaç obert

En aquest cas, al ser llaç obert, solament s'introdueix la consigna. Un cop ja es vol accionar el motor es prem el botó *write*.

En l'etiqueta <button> s'indica que un cop es prem aquest input, ha d'entrar la funció que comença la petició. Aquesta funció s'ha explicat a l'apartat 4.5.3, pas 1.

L'última part es la de la gràfica. La seva configuració és fa tota mitjançant JavaScript. Amb HTML solament es situa i se li assigna un nom per poder referenciar-la des de l'API de Highcharts. Per poder-les distingir a la hora mostrar els valors de lectura, la gràfica de llaç obert tindrà un nom (id) diferent a la del llaç tancat.

4) Estructura llaç tancat. Assignació dels paràmetres i gràfica.

```
<div id="closedloop" class="col s12">
  <div class="row">
    <form class="col s12">
      <div class="row">
        <div class="input-field col s6">
          <input placeholder="Setpoint" id="consignacl" type="number" class="validate">
          <label for="consigna">Consigna</label>
        </div>
        <div class="input-field col s6">
          <input placeholder="Kp" id="kp" type="number" class="validate">
          <label for="Kp">Kp</label>
        </div>
        <div class="input-field col s6">
          <input placeholder="Ki" id="ki" type="number" class="validate">
          <label for="Ki">Ki</label>
        </div>
      </div>
      <div class="row">
        <button id="writecl" onclick="writeValuesClosedLoop()" class="btn waves-effect waves-light" type="button"
          name="action">Write</button>
      </div>
      <div class="row">
          <div id="chartcl">
          </div>
      </div>
    </form>
  </div>

</div>
```

Figura 23. Estructura llaç tancat

4.6.2 Codi CSS

Tal i com es veu a l'apartat anterior, molts elements utilitzen classes. Aquesta propietat ofereix al programador donar-li una forma al element sense haver de descriure l'estil un per un. Bàsicament una classe és un conjunt de estils que es poden referenciar i fer-los servir tantes vegades es vulgui d'una manera ràpida i eficaç. Algunes classes són importades de enllaços externs, però n'hi d'altres que s'han creat al arxiu style.css.

Al codi es podrà apreciar com es descriu la màxima altura, la màxima amplada, els marges en tots els costats del element, la alineació vertical i horitzontal, la disposició del elements, etc.

```css
body {
    margin: 0 auto;
}
.openloop-container{
    width: 100%;
    padding: 25px 0;
    display: flex;
    justify-content: center;
}
.openloop-container > div{
    min-width: 50%;
    text-align: center;
}
.closedloop-container{
    width: 100%;
    padding: 25px 0;
    display: flex;
    justify-content: center;
}
.closedloop-container > div{
    min-width: 50%;
    text-align: center;
}
```

```css
#openloop{
    margin: 25px 0;
}
#closedloop{
    margin: 25px 0;
}
#chart{
    display: flex;
    justify-content: center;
}
```

Figura 24. Codi CSS

4.6.3 Vista general de la web

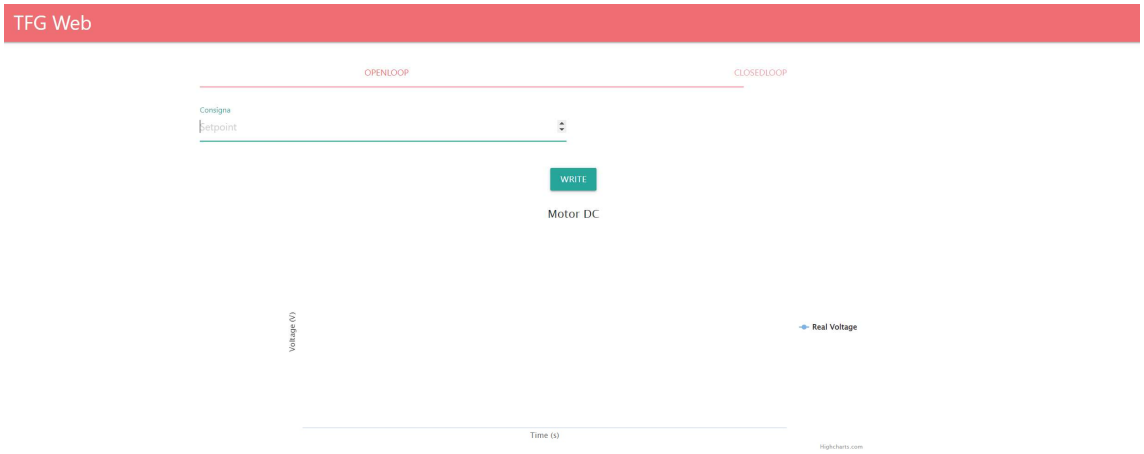

Figura 25. Vista general web

5 Resultats

El cas idoni per comprovar el funcionament complet del projecte seria fent els tests de validacions actuant sobre el motor. Per motius de disponibilitat del laboratori no s'han pogut fer les comprovacions desitjades. Tot i que ja es sap que la comunicació entre la targeta d'adquisició i el motor funciona correctament, hauria estat un test més complet. El que s'ha fet per comprovar tot el software dissenyat ha estat que, en comptes de actuar sobre el motor i agafar la lectura, s'ha agafat la sortida simulada del model. A continuació es mostraran els arxius tant del llaç obert com del llaç tancat.

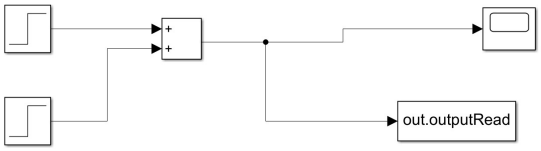

Figura 26. Model prova llaç obert

Amb el llaç obert, el que la web hauria de graficar és el valor del Setpoint introduït. La consigna que hi ha a la part inferior es per retornar la sortida a zero. Un cop feta la prova, la web mostra el següent:

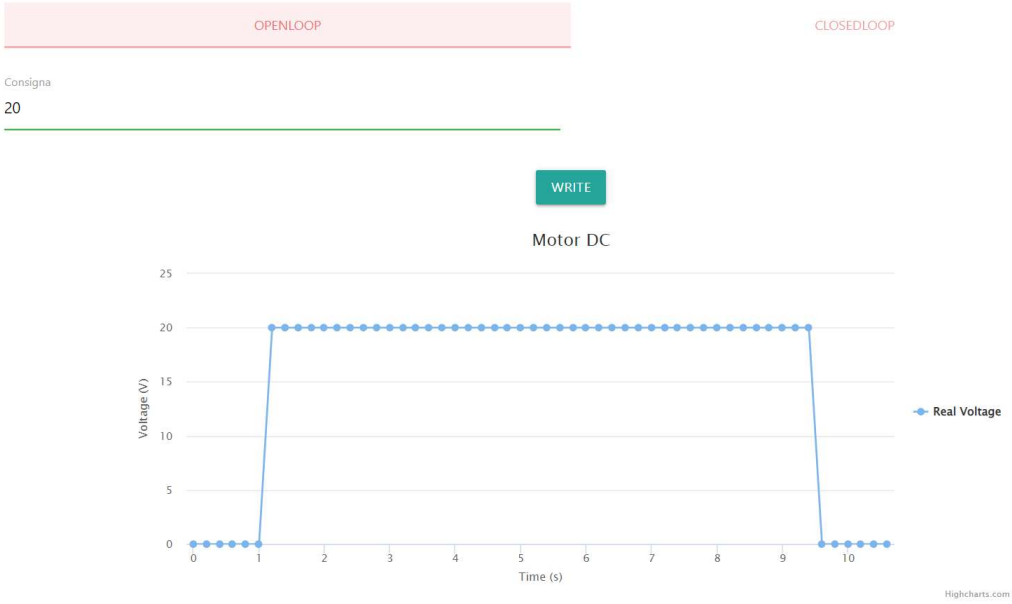

Figura 27. Resultat prova llaç obert

Tal i com es pot observar, la web mostra els 20 Volts de consigna que l'usuari ha introduït. Per tant, amb aquesta prova, el programador s'assegura de que el flux de dades es transmet correctament entre els diferents programes. Al final de la simulació el valor torna a zero ja que així s'indica al arxiu de simulació.

En el cas del llaç tancat, el model incorporarà també el controlador.

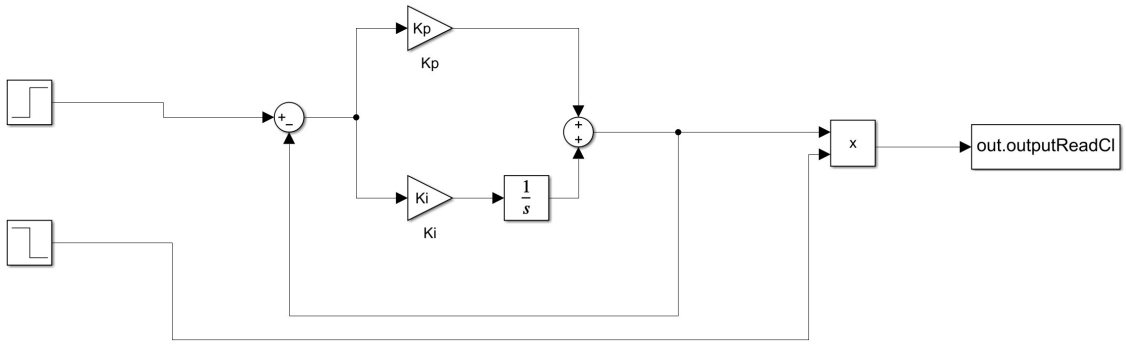

Figura 28. Model prova llaç tacat

En el cas del resultat, la gràfica hauria de mostrar una resposta de una "planta" on no hi haurà cap error, és a dir, el controlador compararà sempre una senyal simulada amb una altra de simulada; no hi motor.

Figura 29. Resultat prova llaç tancat

En llaç tancat també s'aprecia com s'arriba al valor desitjat. El primer ordre que s'aprecia en la transició es deu al controlador.

Un cop finalitzades les probes, ja es pot connectar el motor.

6 Pressupost

En aquest apartat es detallarà el cost per la realització del projecte. Aquest pressupost serà orientatiu ja que, en aquest cas, el material utilitzat ja estava disponible al laboratori. Per tant, no es comptabilitzarà el seu cost.

El cost principal del projecte seran les hores invertides per realitzar-lo. A continuació, es separarà tot el procés en diferents activitats, indicant el cost de cadascuna segons el temps que hagi comportat. El preu per hora que s'establirà serà de 17 euros.

Taula 6. Cost hores invertides

Feina	**Hores (h)**	**Cost (€)**
Recerca per decidir quins llenguatges utilitzar	100	1700
Recerca d'informació per la web	60	1020
Recerca d'informació per la comunicació Python - Matlab	50	850
Realització *front-end* de la web	30	510
Realització *back-end* de la web	40	680
Realització comunicació Python - Matlab	120	2040
Realització models Simulink	20	340
Tests de funcionament del software	60	1020
Tests de funcionament del projecte	40	680
Redacció de la memòria + Reunions per el treball	250	4250
Total hores	**770**	**13090**

L'altre cost que es pot afegir al del les hores invertides és el de la electricitat. Tot el projecte s'ha fet mitjançant un PC per tant, es podria comptabilitzar el seu consum

Temps total = 770h

Consum per hores = 0.275 kW/h

Preu electricitat = 0.24€/kWh

Cost electricitat(€) = 770 * 0.275 * 0.24 =50.82 €

Per tant, el cost total del projecte és de 13090€ + 50.82€ = **13140.82 €**

7 Impacte ambiental

Per acabar d'analitzar la viabilitat del projecte és important fer un estudi de l'impacte ambiental. Com a punt de partida, s'ha de tenir en compte que el projecte sorgeix per millorar un que ja existeix. Per tant, a l'hora d'analitzar-lo es pot fer de dues formes:

- Impacte ambiental del projecte en la seva totalitat.
- Impacte ambiental de la millora del projecte.

En el primer cas, el principal i únic impacte ambiental és el de l'energia. Per poder tindre el motor funcionant i rebent les ordres del PC, aquest conjunt ha d'estar contínuament alimentat. Per tant, aquest consum elèctric comporta un impacte ambiental negatiu ja que, la majoria d'energia generada en Espanya no és renovable.

En el segon cas, si s'analitza solament la millora, s'ha de dir que no s'afegeix cap impacte apart del que ja hi havia. Si s'ha d'afegir algun apunt més, seria que amb la aplicació, l'usuari no s'hauria de moure fins al laboratori per fer funcionar el motor. Amb la web podria controlar el motor des de casa i, d'aquesta manera, evitar la contaminació que comporta el transport fins la zona de treball.

En definitiva, la millora del projecte comporta una millora també en el impacte ambiental respecte al que es tenia anteriorment.

8 Conclusions

El projecte neix amb l'objectiu de poder controlar un motor DC des d'una pàgina web. Des d'aquest punt de partida s'han estudiat totes les vies per portar-ho a terme; llenguatges de programació web, comunicació de la web amb Matlab, adquisició de dades del motor, etc. La part més dificultosa ha estat la de transmetre les dades entre la web i Matlab, això és degut al fet que és una comunicació molt peculiar que no s'acostuma a utilitzar i que, per tant, hi ha molt poca informació. Normalment, els HMI s'acostumen a comunicar amb bases de dades o directament amb Modbus TCP/IP. D'altra banda, el fet de dissenyar la web ha estat bastant més senzill perquè és un concepte molt més genèric i comú.

Per tant, el resultat més rellevant és que s'ha assolit l'objectiu principal. Des d'una interfície web, l'usuari pot enviar la seva consigna i controlar tant en llaç obert com en llaç tancat el motor. El projecte s'ha dissenyat de manera escalable de manera que si en un futur es vol controlar a la vegada diferents motors en un mateix servidor, es podrà fer.

El projecte ha funcionat tal com s'esperava excepte per un motiu. L'ideal seria que quan l'usuari envia la consigna la web solament trigués el temps de simulació per mostrar els valors en la gràfica. La realitat és que el temps és molt major. Això és deu al fet que, quan s'executa la petició, Matlab i Simulink s'han d'obrir i executar. Aquest procés, depenent de l'ordinador pot ser molt lent; sobretot les primeres peticions. Un cop, aquests programes ja estan funcionant en segon pla, la lectura és molt més ràpida i més semblant al que s'esperava en un principi.

Per tant, si s'ha de millorar el projecte en un futur, es podria automatitzar de manera que els programes s'iniciessin un cop entres a la web, i que no s'esperi fins a enviar la consigna com fa ara. També es podria afegir una *dashboard* amb tots els motors del laboratori i així, poder escollir el que està lliure o sense funcionar. Finalment, el fet d'allotjar la web en un domini, en comptes de fer-ho en un servidor local com ara, farà que et puguis comunicar remotament.

En definitiva, és un projecte on s'ha pogut aprendre tant del món de la programació web, com del món del control i simulacions. I aquests eren els objectius principals.

9 Referències

angular. (27 / 04 / 2022). Recollit de angular: https://angular.io/

cursosaula21. (07 / 03 / 2022). Recollit de cursosaula21: https://www.cursosaula21.com/que-es-un-sistema-scada/

Degiovannini, M. (12 / 2005). *Comparativa de Frameworks WEB.* Recollit de Comparativa de Frameworks WEB: http://static1.1.sqspcdn.com/static/f/923743/15025206/1320739503647/frameworks_web.pdf?token=0p0jLXicjEHOLxPkNFSBDnZXMYo%3D

developer.mozilla. (05 / 04 / 2022). Recollit de developer.mozilla: https://developer.mozilla.org/es/

expressjs. (29 / 05 / 2022). Recollit de expressjs: https://expressjs.com/

geeksforgeeks. (05 / 04 / 2022). Recollit de geeksforgeeks: https://www.geeksforgeeks.org/

ifgeekthen.nttdata. (29 / 05 / 2022). Recollit de ifgeekthen.nttdata: https://ifgeekthen.nttdata.com/es/que-es-expressjs-y-primeros-pasos

mathworks. (20 / 05 / 2022). Recollit de mathworks: https://es.mathworks.com/help/matlab/matlab-engine-for-python.html

nodejs. (02 / 06 / 2022). Recollit de nodejs: https://nodejs.org/api/child_process.html#child_processspawncommand-args-options

Presta, M. (10 / 05 / 2022). *back4app*. Recollit de back4app: https://blog.back4app.com/es/los-diez-mejores-lenguajes-de-backend/

Raghvendra, N. (15 / 03 / 2022). *SCADA System – Components, Hardware & Software Architecture, Types*. Recollit de SCADA System – Components, Hardware & Software Architecture, Types: https://electricalfundablog.com/scada-system-components-architecture/

reactjs. (27 / 04 / 2022). Recollit de reactjs: https://es.reactjs.org/

vuejs. (27 / 04 / 2022). Recollit de vuejs: https://vuejs.org/

10 Annexos

A part d'adjuntar el codi a la memòria, també s'ha creat un repositori on es podran trobar tots els arxius necessaris:

https://github.com/achrafelyahyati/TFGWeb

Codi complet:

HTML

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head><!--save all fonts and styles needed-->
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>TFGWeb</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;700&di
splay=swap" rel="stylesheet">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materi
alize.min.css">
    <link rel="stylesheet" type="text/css" href="/css/style.css">
</head>
<body>
    <nav>
        <div class="nav-wrapper">
          <a href="#" class="brand-logo" style="margin-left: 50px">TFG
Web</a>
          <!--ul id="nav-mobile" class="right hide-on-med-and-down">
            <li><a onclick="M.toast({html: 'Connected!'})" class="waves-
effect waves-light btn">CONNECT <i class="material-icons
right">arrow_forward</i></a></li>
          </ul-->
        </div>
    </nav>
    <section class="openloop-container">
        <div class="row">
            <div class="col s12">
              <ul class="tabs">
```

```
                <li class="tab col s6"><a class="active"
href="#openloop">OPENLOOP</a></li>
                <li class="tab col s6"><a
href="#closedloop">CLOSEDLOOP</a></li>
              </ul>
            </div>
            <div id="openloop" class="col s12">
                <div class="row">
                    <form class="col s12">
                      <div class="row">
                        <div class="input-field col s6">
                          <input placeholder="Setpoint" id="consignaop"
type="number" class="validate">
                          <label for="consigna">Consigna</label>
                        </div>
                      </div>
                      <div class="row">
                        <button id="writeop"
onclick="writeValuesOpenLoop()" class="btn waves-effect waves-light"
type="button" name="action">Write</button>
                      </div>
                      <div class="row">
                          <div id="chart">

                          </div>
                      </div>
                    </form>
                  </div>
            </div>
            <div id="closedloop" class="col s12">
              <div class="row">
                <form class="col s12">
                  <div class="row">
                    <div class="input-field col s6">
                      <input placeholder="Setpoint" id="consignacl"
type="number" class="validate">
                      <label for="consigna">Consigna</label>
                    </div>
                    <div class="input-field col s6">
                      <input placeholder="Kp" id="kp" type="number"
class="validate">
                      <label for="Kp">Kp</label>
                    </div>
                    <div class="input-field col s6">
                      <input placeholder="Ki" id="ki" type="number"
class="validate">
                      <label for="Ki">Ki</label>
                    </div>
                  </div>
```

```
                    <div class="row">
                      <button id="writecl"
onclick="writeValuesClosedLoop()" class="btn waves-effect waves-light"
type="button" name="action">Write</button>
                    </div>
                    <div class="row">
                        <div id="chartcl">
                        </div>
                    </div>
                  </form>
                </div>

              </div>
            </div>
    </section>


    <!-- scripts -->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/material
ize.min.js"></script>
    <script defer src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
    <script defer
src="https://code.highcharts.com/highcharts.js"></script>
    <script defer src="/script.js"></script>
</body>
</html>
```

CSS

Style.css

```
body {
    margin: 0 auto;
}
.openloop-container{
    width: 100%;
    padding: 25px 0;
    display: flex;
    justify-content: center;
}
.openloop-container > div{
    min-width: 50%;
    text-align: center;
}
.closedloop-container{
    width: 100%;
    padding: 25px 0;
    display: flex;
    justify-content: center;
}
.closedloop-container > div{
    min-width: 50%;
    text-align: center;
}
#openloop{
    margin: 25px 0;
}
#closedloop{
    margin: 25px 0;
}
#chart{
    display: flex;
    justify-content: center;
}
```

JavaScript

Script.js

```
//var REC = require('require.js')


$(document).ready(function(){
    M.AutoInit();
    console.log("FLAG")
    runChart()
    runChartCl()

});

let runChart = (data = []) => {
    Highcharts.chart('chart', {

        title: {
            text: 'Motor DC'
        },

        subtitle: {
            text: ''
        },

        yAxis: {
            title: {
                text: 'Voltage (V)'
            }
        },

        xAxis: {

            tickInterval: 1,
            title: {
                text: 'Time (s)'
            }
        },

        legend: {
            layout: 'vertical',
            align: 'right',
            verticalAlign: 'middle'
        },

        plotOptions: {
            series: {
                label: {
```

```
                connectorAllowed: false
            },
            pointStart: 0,
            pointInterval: 0.2
        }
    },

    series: [{
        name: 'Real Voltage',
        data: data
    }],

    responsive: {
        rules: [{
            condition: {
                maxWidth: 500
            },
            chartOptions: {
                legend: {
                    layout: 'horizontal',
                    align: 'center',
                    verticalAlign: 'bottom'
                }
            }
        }]
    }

});
}

let runChartCl = (data = []) => {
    Highcharts.chart('chartcl', {

    title: {
        text: 'Motor DC'
    },

    subtitle: {
        text: ''
    },

    yAxis: {
        title: {
            text: 'Voltage (V)'
        }
    },

    xAxis: {
        tickInterval: 1,
```

```
            title: {
                text: 'Time (s)'
            }
        },

        legend: {
            layout: 'vertical',
            align: 'right',
            verticalAlign: 'middle'
        },

        plotOptions: {
            series: {
                label: {
                    connectorAllowed: false
                },
                pointStart: 0,
                pointInterval: 0.2
            }
        },

        series: [{
            name: 'Real Voltage',
            data: data
        }],

        responsive: {
            rules: [{
                condition: {
                    maxWidth: 500
                },
                chartOptions: {
                    legend: {
                        layout: 'horizontal',
                        align: 'center',
                        verticalAlign: 'bottom'
                    }
                }
            }]
        }

    });
}

function writeValuesOpenLoop(){

    var opSetpoint =  $('#consignaop').val() ? $('#consignaop').val() :
null;
    if(opSetpoint === null || opSetpoint<0){
```

```
        alert("Setpoint not assigned")
    }
    requestWriteOpenloopPy(opSetpoint);

}

function requestWriteOpenloopPy(opSetpointPy){

    url = 'http://localhost:3000/receive?time=' + opSetpointPy + ''
      $.ajax({
        type:"GET", // la variable type guarda el tipo de la peticion
GET,POST,..
        url:url, //url guarda la ruta hacia donde se hace la peticion
        success:function(datos, status){ //success es una funcion que se
utiliza si el servidor retorna informacion
             console.log({datos, status})
             datos = JSON.parse(datos)
             var datosCleaned = []
             for (let i = 0; i < datos.length; i++) {
                 const dato = datos[i];
                 for (let j = 0; j < dato.length; j++) {
                     const val = dato[j];
                     datosCleaned.push(val)
                 }
             }
             runChart(datosCleaned)
         },
    })
}

function writeValuesClosedLoop(){

    var clSetpoint =  $('#consignacl').val() ? $('#consignacl').val() :
null;
    var clKp =  $('#kp').val() ? $('#kp').val() : null;
    var clKi =  $('#ki').val() ? $('#ki').val() : null;

    if(clSetpoint === null || clSetpoint<0){
        alert("Setpoint not assigned")
    }
    if(clKp === null || clKp<0){
        alert("Kp not assigned")
    }
    if(clKi === null || clKi<0){
        alert("Ki not assigned")
    }
    clSetpoint = parseFloat(clSetpoint);
    clKp = parseFloat(clKp);
    clKi = parseFloat(clKi);
```

```
    requestWriteClosedloopPy(clSetpoint, clKp, clKi);
}


function requestWriteClosedloopPy(clSetpointPy, clKpPy, clKiPy){


    console.log(typeof(clSetpointPy));
    var arraycl = [clSetpointPy, clKpPy, clKiPy]
    console.log(arraycl);
    console.log(typeof(arraycl));
    url = 'http://localhost:3000/receivecl?time=' + arraycl + ''
    $.ajax({
        type:"GET",
        url:url,
        success:function(datos, status){
             console.log({datos})
             datos = JSON.parse(datos)
             var datosCleaned = []
             for (let i = 0; i < datos.length; i++) {
                 const dato = datos[i];
                 for (let j = 0; j < dato.length; j++) {
                     const val = dato[j];
                     datosCleaned.push(val)
                 }
             }
             runChartCl(datosCleaned)
         },
    })
}
```

Server.js

```
const express = require('express')
const app = express()
const port = 3000
const path = require('path');

app.use(express.static(__dirname + '/public'));
app.use(express.static(__dirname + '/client'));

app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, '/client/index.html'));
})

app.get('/receive', (req, res) => {


    var Setpoint = req.query.time;
    const spawner = require('child_process').spawn;

    console.log('Data sent to python script: ', Setpoint);
    const python_process = spawner('python',['./connectMatlab.py',
Setpoint]);

    python_process.stdout.on('data', (data) =>{
       var arrayRead = data
       console.log('Data received from python script open loop:',
data.toString());
       res.send(arrayRead)
      // console.log('Data received from python script cleaned:', data);
    });


})

app.get('/receivecl', (req, res) => {

  var arrayClPy = req.query.time.split(',');
  const spawner = require('child_process').spawn;

  console.log('Data sent to python script: ', arrayClPy);

  const python_processCl = spawner('python',['./clconnectMatlab.py',
arrayClPy]);
  //console.log(python_processCl);
  python_processCl.stdout.on('data', (dataCl) =>{
     var arrayReadCl = dataCl;
     console.log('Data received from python script closed loop:',
arrayReadCl.toString());
```

```
    res.send(arrayReadCl);
    // console.log('Data received from python script cleaned:', data);
  });


})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

ConnectMatlab.py

```
import sys                                          #node.js connectivity
import matlab.engine                                #matlab connectivity
setpointPy = float(sys.argv[1])                     #we get the
simulation/setpoint time from the server
matlabEngine = matlab.engine.start_matlab()         #start matlab engine
matlabEngine.init(nargout=0)

matlabEngine.workspace['Setpoint'] = setpointPy       #important to send
as float!!!!
matlabEngine.workspace['InvSetpoint'] = -setpointPy

matlabEngine.simRead(nargout=0)
plotArrayRead = matlabEngine.workspace['outputRead']


print(matlabEngine.workspace['outputRead'])
```

ClConnectMatlab.py

```
import sys                                          #node.js connectivity
import matlab.engine                                #matlab connectivity

arrayPy = sys.argv[1].split(',')

matlabEngine = matlab.engine.start_matlab()         #start matlab engine
matlabEngine.initcl(nargout=0)

matlabEngine.workspace['SetpointCl'] = float(arrayPy[0])       #important
to send as float!
matlabEngine.workspace['InvSetpointCl'] = -float(arrayPy[0])
matlabEngine.workspace['Kp'] = float(arrayPy[1])
matlabEngine.workspace['Ki'] = float(arrayPy[2])

matlabEngine.simReadCl(nargout=0)

print(matlabEngine.workspace['outputReadCl'])
```