FACULTAT
DE CIÈNCIES, TECNOLOGIA
I ENGINYERIES
UVIC | UVIC·UCC

Treball de Fi de Grau Experimental

DISSENY I CONSTRUCCIÓ D'UN MECANISME SIMULADOR DEL MOVIMENT OSCIL·LATORI

GUILLEM MAZA CUTRINA

Grau en enginyeria mecatrònica

Tutor/a: Moisès Serra Serra

Co-tutor Moises Garín Escrivá

Ubicació Vic, juny de 2022

Agraïments

Dono les gràcies, en primer lloc, als tutors per tot el suport rebut al llarg del projecte, per motivar-me en els moments més complicats, per mostrar un gran interès amb el treball, per la seva predisposició a trobar-nos per solucionar dubtes i per ensenyar-me eines que, sense dubte, seran útils pel meu futur com a enginyer.

De la mateixa manera, agraeixo a en Jordi Serra i a la universitat per subministrar-me el material necessari per construir el mecanisme i fer ús de les eines de les quals es disposen a les instal·lacions.

Tanmateix, vull agrair a l'Art de la Fusta Mecanitzats per deixar-me fer ús de les seves instal·lacions i màquines per poder fer algun procés de mecanitzat necessari per a la construcció del mecanisme.

Finalment, agraeixo i valoro molt positivament als meus companys del grau i familiars més pròxims per fer més amenes les hores d'estudi malgrat els moments tensos viscuts.

Resum

Títol: *Disseny i construcció d'un mecanisme simulador del moviment oscil·latori.*
Autor: Guillem Maza Cutrina.
Co-Tutors: Moisès Serra Serra i Moises Garín Escrivá.
Data: juny de 2022
Paraules clau: *molla, esmorteïdor, massa, disseny, programació, construcció, Arduino, Simulink.*

Els mecanismes formats per masses, molles i esmorteïdors són parts molt importants per garantir l'estabilitat, tracció i la seguretat de gran part dels vehicles amb els quals ens movem pràcticament cada dia. Imaginar una oscil·lació no és una tasca molt complexa, però sí que és una mica més complex saber com evoluciona aquesta oscil·lació en canviar el valor de la massa a la qual està sotmès, la constant de la molla, la constant d'esmorteïment i la posició inicial a la qual s'inicia aquest moviment oscil·latori. És per això, que hi ha implementats softwares que permeten fer aquest tipus de simulacions, però per entendre bé els resultats obtinguts amb aquests programes és important veure físicament com evolucionen aquestes oscil·lacions. Per aquest motiu, s'ha desenvolupat aquest treball que consta del disseny, la programació i la fabricació d'un mecanisme que permeti veure i entendre el moviment oscil·latori en el qual l'usuari escull la massa que es pretén simular, la posició inicial del sistema, la constant de la molla i la constant d'esmorteïment.

El procés comença amb els fonaments de l'obtenció d'un model matemàtic per poder controlar en cada instant de temps la posició a la qual es troba el sistema en funció dels paràmetres desitjats per l'usuari. Se segueix amb la realització d'un model mecatrònic, que consisteix a fer el disseny complet del prototip, tenint en compte tant les parts mecàniques com les parts electròniques. És durant aquest procés de disseny que es realitza, paral·lelament, la cerca i selecció dels components electrònics necessaris. Una vegada conegut el disseny del mecanisme, s'inicia el procés de construcció i de programació, conjuntament amb la comprovació del funcionament del sistema.

Summary

Title: *Design and construction of an oscillatory motion simulator mechanism.*
Author: Guillem Maza Cutrina.
Supervisor: Moisès Serra Serra and Moises Garín Escrivá.
Date: June 2022
Keywords: *spring, shock absorber, mass, design, programming, building, Arduino, Simulink.*

The mechanisms made up of masses, springs and shock absorbers are very important parts to ensure the stability, traction and safety of many vehicles with we move with almost every day. Imagining an oscillation is not a very complex task, but it is little more complex to know how this oscillation evolves by changing the value of the mass, the spring constant, the damping constant and the initial position where this oscillating motion begins. It is for that reason that software has been implemented to carry out these types of simulations, but in order to fully understand the results obtained with these programs, it is important to physically see how these oscillations evolve. That is why this project has been developed which consists in designing, programming and manufacturing a mechanism that allows to see and understand the movement that follows these types of mechanisms where the user chooses the mass to be simulated, the initial position of the system, the spring constant and the damping constant.

The process begins with the basics of obtaining a mathematical model to be able to control at each instant the position of the system according to the parameters desired by the user. It is followed by the realization of a mechatronic model, which consists of making the complete design of the prototype, considering the mechanical parts and the electronic parts. It is during this design process that the search and selection of the necessary electronic devices is carried out in parallel. Once the design of the mechanism is known, the construction and programming process will be carried out together with the verification of the operation of the system.

Índex de continguts

Índex de figures

Índex de taules

Índex de figures de l'apèndix

Glossari

Matlab. Segons MathWorks, és un programari multiplataforma amb un entorn de computació numèrica i un llenguatge de programació dissenyat per Cleve Moler.
PTC Creo Parametric. És un programari que permet el disseny i anàlisis de peces o ve conjunts mecànics.
Solidworks. És un programa de disseny assistit per ordinador de modelatge sòlid i enginyeria assistida per ordinador publicada per Dassault Systèmes.
Arduino IDE. És un programari lliure de llicències que permet de forma senzilla l'escriptura de codi i enviament d'aquest a qualsevol placa Arduino.
Simulink. És un entorn de diagrames de blocs que s'utilitza per dissenyar sistemes amb models en diferents dominis i simular sistemes.
Hiwin. Es tracta d'una marca comercial la qual destaca per la fabricació de guies i patins amb un coeficient de lliscament molt baix.
App designer. És una extensió de Matlab que permet crear aplicacions sense que sigui necessari ser un desenvolupador de programari professional.

Acrònims

PLA. Àcid polilàctic, molt utilitzat en impressió 3D.

PWM. Pulse-Width Modulation, en català conegut com a modulació per amplada de polsos.

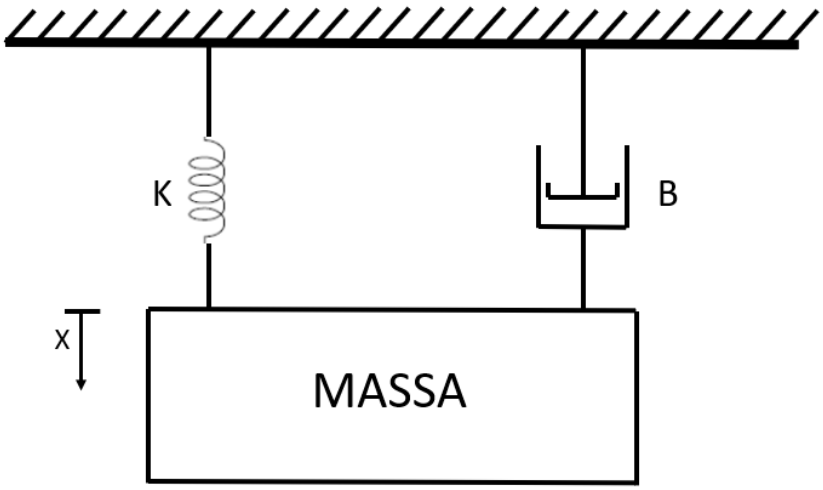
CAD. Computer-Aided Design, en català traduït com a disseny assistit per ordinador.

CAE. Computer-Aided Engineering, en català traduït com enginyeria assistida per ordinador.

USB. Universal Serial Bus, en català traduït com a bus universal en sèrie.

1. Introducció

Aquesta memòria exposa l'explicació dels passos que s'han seguit per assolir el disseny, construcció i programació d'un sistema de simulació de les oscil·lacions format per una molla, un esmorteïdor i una massa. Per entendre una mica millor el sistema que es vol simular, a continuació es mostra una representació esquemàtica.



K: constant elàstica de la molla
B: coeficient de viscositat amortidor
X: correspon al desplaçament

Figura 1 *Representació del sistema*

Tal com es pot apreciar, hi ha una molla, amb la seva constant elàstica K, un esmorteïdor amb el coeficient de viscositat B i un bloc que representa la massa la qual es penja de la molla i de l'esmorteïdor.

S'ha dut a terme aquest projecte, ja que a vegades és una mica complicat imaginar-se com canvien les oscil·lacions d'un mecanisme massa-molla modificant la constant elàstica de la molla, la massa que se simula i la posició inicial de l'oscil·lació, afegint també, l'escassa existència de mecanismes que permetin veure físicament aquestes oscil·lacions. Per això es pretén que aquest projecte acabi sent un recurs pedagògic per futurs alumnes. És justament per aquest motiu que s'ha dut a terme aquest projecte, per entendre amb major facilitat com varien les oscil·lacions en funció de diferents paràmetres que intervenen en el moviment.

La forma en la qual es representarà les oscil·lacions és mitjançant el mecanisme biela-manovella, tal com el tutor del projecte ha demanat com a requisit, de la mateixa manera que s'han de poder simular oscil·lacions amb una amplitud al voltant de 0.2 metres amb una freqüència de fins a aproximadament 1 hertz.

Aquest projecte s'estructura en tres parts, la part mecànica, la part electrònica i la programació. La part mecànica consisteix a escollir els materials que s'utilitzaran per a l'estructura i en fer el disseny del mecanisme per tal de poder construir-lo a posteriori. En conseqüència, en aquesta part es farà ús del PTC Creo Parametric i del SolidWorks per fer el disseny de cada una de les parts, fer l'assemblatge i simulacions de l'estructura. A més, el procés de fabricació i construcció del mecanisme d'acord amb el disseny fet prèviament també forma part d'aquest segment. La part de l'electrònica té a veure amb escollir els components electrònics adients per fer funcionar correctament el mecanisme de simulació d'acord amb les restriccions mecàniques d'aquest. En l'apartat de programació s'engloba la programació de la placa controladora per així, seguint una sèrie de càlculs, poder controlar els components que faran possible la simulació de les oscil·lacions. Aquesta part recull també l'elaboració d'una aplicació d'ordinador per entrar els paràmetres principals que caracteritzen el moviment oscil·latori d'aquest tipus.

Així que posant en comú les parts que formen el projecte, es pretén controlar el moviment del mecanisme establint una comunicació en temps real mitjançant el port sèrie entre un ordinador, que conté l'aplicació d'entrada de dades, la placa Arduino i l'actuador encarregat de moure el mecanisme, seguint l'esquema gràfic que es pot apreciar a continuació.

Figura 2 *Esquema gràfic funcionament*

1.1. Objectius

L'objectiu general i principal del projecte és dissenyar i construir un mecanisme que permeti simular el moviment oscil·latori de les molles per complir les restriccions de la constant de la molla, la constant d'esmorteïment, la distància d'oscil·lació i la massa que se simula. Per fer-ho és necessari complir els següents objectius secundaris:

- Fer un mecanisme segur que sigui un recurs pedagògic per ensenyar, aprendre i veure el moviment oscil·latori de les molles de forma clara i entenedora.
- Utilitzar el mecanisme biela-manovella per representar el moviment del sistema.
- Fer una aplicació simple, visual i entenedora per entrar dades.
- Aconseguir la modelització matemàtica de les equacions del moviment en el domini freqüencial i en el domini temporal per treballar amb el Simulink i l'Arduino respectivament.
- Fer el sistema accessible per a la majoria de gent i, per tant, programar amb softwares lliures de llicència de pagament.
- Dissenyar, mecanitzar i construir un prototip lleuger, de menys de 10 quilograms, robust i manejable.
- Construir un mecanisme amb un cost inferior a 1000 €.
- Complir els requeriments d'amplitud d'oscil·lacions de més de 0.2 metres i de freqüències d'aproximadament 1 hertz aconseguint, d'aquesta manera, flexibilitat per poder simular diferents casos.

1.2. Estructura de la memòria

Aquesta memòria segueix un ordre específic, organitzant la informació amb capítols, per tal de poder dur a terme un projecte mecatrònic com el que s'ha dut a terme. Així que s'ha començat pels aspectes més teòrics i acabat amb la part més pràctica del projecte. En la primera part s'expliquen aspectes més teòrics per tal de conèixer com funciona el sistema i descobrir els possibles camins a seguir pel desenvolupament del projecte. Finalment, els últims capítols fan referència a aspectes concrets que s'han desenvolupat per dur a terme el mecanisme. A continuació es pot veure resumit en forma de punts què és el què s'explica en cada un d'aquests capítols.

- El capítol 2 recull l'estat de l'art en el qual s'explica com funciona un sistema biela-manovella, un sistema format per una molla i un esmorteïdor conjuntament amb alternatives del mecanisme utilitzat.
- El capítol 3 detalla com s'han obtingut les equacions que descriuen el model matemàtic del sistema.
- En el capítol 4 mostra com s'ha desenvolupat el projecte tant pel que fa a aspectes mecànics, com seria disseny i construcció, com a aspectes més electrònics com seria escollir components electrònics i la programació.
- El capítol 5 explica el cost que ha tingut dur a terme aquest projecte.
- El capítol 6 recull el manual d'instruccions d'ús, en el qual s'exposen els procediments que cal seguir per usar correctament el mecanisme.
- El capítol 7 mostra la discussió dels resultats i conclusions conjuntament amb els aspectes a millorar del projecte.

A més, s'ha complementat la memòria amb els següents apèndixs:

- En l'apèndix A es mostra els plànols de les peces dissenyades.
- En l'apèndix B es presenta el codi de programació de l'Arduino IDE.
- En l'apèndix C es recull el codi de programació de l'aplicació.
- En l'apèndix D es mostra l'esquema de connexions.

2. Estat de l'art

Les simulacions dels sistemes que contenen molles i esmorteïdors, principalment, es fan en forma de simulacions mitjançant software CAE (de l'anglès *Computer Aided Engineering*), els quals permeten comprovar la resposta d'un mecanisme prèviament dissenyat en imposar-hi esforços o càrregues. El fet que mitjançant el software es pugui simular el funcionament d'un sistema molla-esmorteïdor amb infinitat de combinacions de massa, constant de la molla i constant de viscositat de l'esmorteïdor fa que l'existència de mecanismes físics per simular el funcionament sigui pràcticament inexistent. Així que tal com s'ha vist en els objectius definits anteriorment, el que es pretén és fer un sistema que simuli aquest moviment fent ús del sistema biela-manovella.

2.1. Funcionament sistema biela-manovella

Biela-manovella és el sistema per excel·lència per transformar el moviment circular o rotacional a un moviment lineal i a la inversa. Principalment, el sistema consta de les dues parts que es troben unides per una articulació:

- Biela: és un element rígid encarregat d'unir la manovella amb l'element que ha de definir el moviment lineal, tal com podria ser un patí.
- Manovella: és un element rígid el qual per un dels seus extrems es connecta amb la biela i per l'altre crea un punt de rotació pura en subjectar-se o en connectar-se a un motor.

A continuació es pot veure un exemple de sistema biela-manovella on l'element negre correspon a la manovella i el braç vermell a la biela.

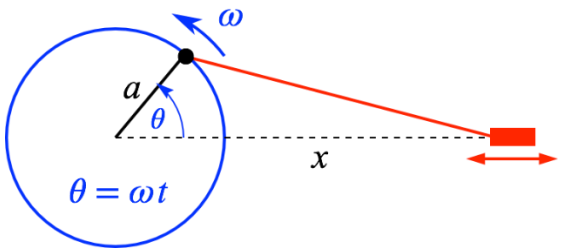


Figura 3 *Mecanisme biela-manovella. Extret de "An Example of a Non-Harmonic Oscillator: Crankshaft Motion", Russell, D. Llicència CC 4.0.*

En aquest projecte es pretén transformar el moviment circular d'un motor en un moviment lineal, ja que el motor serà l'encarregat de proporcionar l'energia per moure el mecanisme. Per tant, es connectarà un dels extrems de la manovella al motor i l'altre extrem de la manovella a la biela, permetent d'aquesta forma realitzar el moviment.

2.2. Funcionament sistema molla-esmorteïdor

El sistema molla-esmorteïdor és un dels sistemes més utilitzats en el món de l'automoció amb la funció principal d'absorbir les irregularitats del terreny i amb l'objectiu d'augmentar la comoditat i el control del vehicle.

Una molla és un element mecànic elàstic que permet acumular i desprendre energia sense que es deformi plàsticament, sempre que no se superi el límit de la constant elàstica de la molla. Per tant, el que permet la molla és crear oscil·lacions mitjançant la seva compressió i extensió, les quals es van atenuant a causa de la fricció, temperatura, etc. Hi ha diferents tipus de molles en funció de les seves aplicacions, uns exemples de molles són les molles de tracció, de compressió, de torsió, d'espiral, de ballesta, etc.

Un esmorteïdor o també conegut com a amortidor, en canvi, és un element que la seva funció principal és absorbir les energies d'un impacte, convertint l'energia cinètica obtinguda pel moviment causat pel xoc, en un altre tipus d'energia, principalment en forma de calor. Destacadament, hi ha esmorteïdors de gas, hidràulics, pneumàtics, etc.

Per tant, una vegada definides les funcions de cada un dels elements que formen el sistema molla-esmorteïdor, la funció que fa la molla és principalment suavitzar una irregularitat fent oscil·lacions, per evitar que l'impacte sigui brusc mentre que la funció de l'esmorteïdor és frenar les oscil·lacions, és a dir, evitar que el nombre d'oscil·lacions sigui molt gran, tot depenent de la constant de la molla i la constant de l'esmorteïdor. A continuació es pot veure una imatge d'aquest sistema.

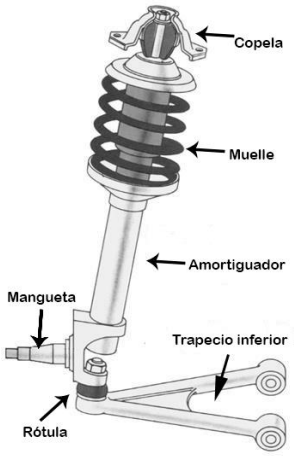


Figura 4 *Sistema molla-esmorteïdor. Extret de "Sistema de suspensión", GETAUTO. Copyright 2017.*

2.3. Alternatives al mecanisme

Una alternativa que també es va estudiar per representar el moviment lineal és mitjançant l'accionament d'un vis sense fi el qual mou un suport al llarg de la seva rosca. Aquest sistema permet de forma més simplificada passar d'un moviment circular a un moviment lineal. A continuació es pot veure un exemple.

Figura 5 *Mecanisme moviment lineal amb vis sense fi. Extret de "Guía lineal de alta precisión", Alibaba. Copyright.*

Tal com es pot veure en la Figura 5, el moviment de la part sòlida és efectuat per la rotació del vis sense fi.

3. Modelització matemàtica

Assolir un model matemàtic del moviment oscil·latori és molt important per tal d'aconseguir un control ràpid i precís.

3.1. Obtenció de les equacions del moviment en el domini freqüencial

Amb la modelització matemàtica es pretén, mitjançant la teoria de Katsuhiko Ogata (2010), obtenir les equacions necessàries per representar el moviment real.

A partir de la representació esquemàtica del mecanisme mostrat a la Figura 1, és més senzill escriure les equacions que definiran el moviment del mecanisme. Com que només hi ha una massa, amb una única equació en tindrem prou, ja que s'han d'escriure tantes equacions com masses hi ha al sistema (Ogata, 2010).

Una vegada mostrat i escollit el sentit positiu, cal representar les forces que actuen i, amb el sentit correcte. A continuació es pot veure l'esquematització del sistema amb les forces i el sentit que es considera positiu, concretament avall.

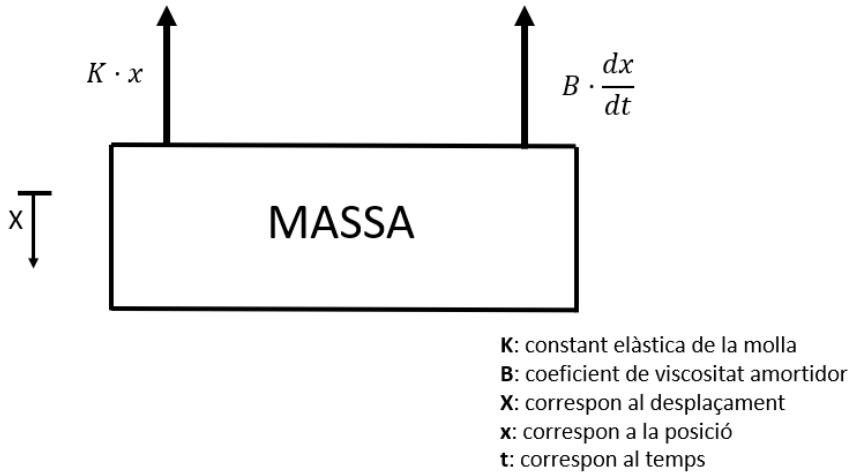


Figura 6 *Representació esquemàtica de les forces*

Tenint en compte la segona llei de Newton:

$$\Sigma F = m \cdot a \qquad (1)$$

Es pot escriure l'equació del sistema de la següent forma, tenint en compte que s'ha expressat la velocitat com la derivada primera de la posició respecte del temps i l'acceleració, com la derivada segona de la posició respecte del temps.

$$-K \cdot X(t) - B \cdot \frac{dX(t)}{dt} = m \cdot \frac{d^2X(t)}{dt}$$

(2)

És important observar que l'anterior equació es tracta d'una equació diferencial d'ordre 2, ja que involucra la derivada segona de la variable de la posició.

Per facilitar l'anàlisi del comportament del sistema, el següent que es fa amb l'equació *(2)* és fer *La Transformada de Laplace,* ja que aïllar la posició no és del tot fàcil. *La Transformada de Laplace* permet passar l'equació del domini del temps al domini de la freqüència mitjançant la seva definició. A continuació es pot veure la definició de *La Transformada de Laplace*:

$$\mathcal{L}[f(t)] = F(s) = \int_{0-}^{\infty} f(t)e^{-st}dt$$

(3)

Així que *La Transformada de Laplace* de l'equació que descriu el moviment del sistema queda de la següent forma:

$$-K \cdot X(s) - B \cdot s \cdot X(s) = m \cdot s^2 \cdot X(s)$$

(4)

A continuació cal ordenar i aïllar la posició en el domini de la freqüència tal com es pot veure en la següent equació.

$$X(s) = \frac{1}{m} \cdot [\frac{1}{s} \cdot (-B \cdot X(s) - \frac{1}{s} \cdot (K \cdot X(s)))]$$

(5)

La Figura 7 mostra l'equació anterior (5) implementada amb el Simulink de Matlab representada en forma de blocs.

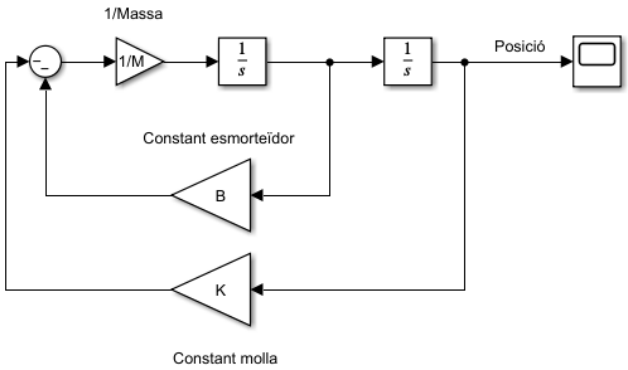


Figura 7 *Representació amb Simulink de l'equació (5)*

3.2. Obtenció de les equacions del moviment en el domini temporal

Per fer més fàcil els càlculs de la posició i poder programar amb l'Arduino s'han obtingut, també, les equacions en el domini del temps. Fer els càlculs en el domini temporal redueix el temps de càlcul, ja que són càlculs més simples i, per tant, es redueix els requeriments de capacitat de càlcul pel dispositiu utilitzat per programar.

Per aconseguir aquestes equacions, s'ha partit de l'esquema que es pot veure a la Figura 8 que hi ha indicat en quin punt hi ha l'acceleració, la velocitat i la posició.

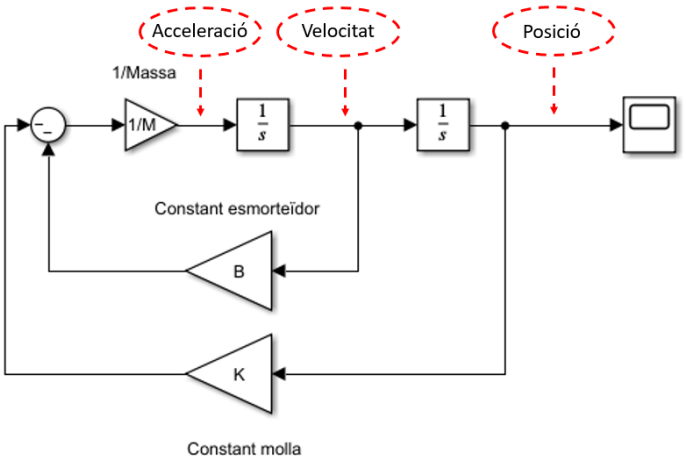


Figura 8 *Representació de l'acceleració, velocitat i posició*

Per tant, si es busca l'acceleració només cal completar el següent enllaç tancat tenint en compte que la velocitat és dx i la posició és x. La Figura 9 mostra l'esquema que simplifica l'obtenció de les equacions.

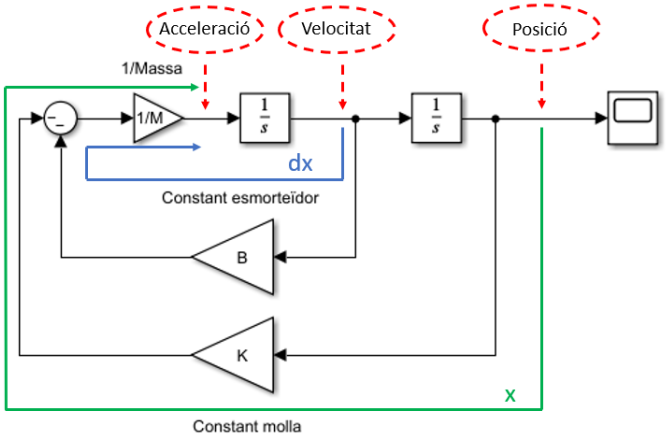


Figura 9 *Representació esquemàtica de l'obtenció de les equacions*

D'aquesta manera es poden escriure les equacions de l'acceleració, la velocitat com l'integral de l'acceleració i la posició com l'integral de la velocitat.

$$Acc = \frac{(-K \cdot x - B \cdot dx)}{M} \tag{6}$$

$$V = Acc \cdot dt \tag{7}$$

$$X = V \cdot dt \tag{8}$$

3.3. Control de l'angle girat en funció de la posició

Les equacions del moviment explicades en l'apartat anterior permeten conèixer en cada moment la posició de la massa partint d'un zero conegut, però el mecanisme que s'utilitza per representar el moviment no en té suficient de conèixer la posició de la massa sinó que li és necessari conèixer l'angle que ha recorregut la manovella. Així que a continuació es detalla com s'obté l'angle girat en funció de la posició de la molla en cada instant.

Al cap i a la fi, el mecanisme biela-manovella acaba tancant un triangle en la majoria de posicions, així que mitjançant el teorema del cosinus es pot resoldre l'angle tal com es pot veure a continuació.

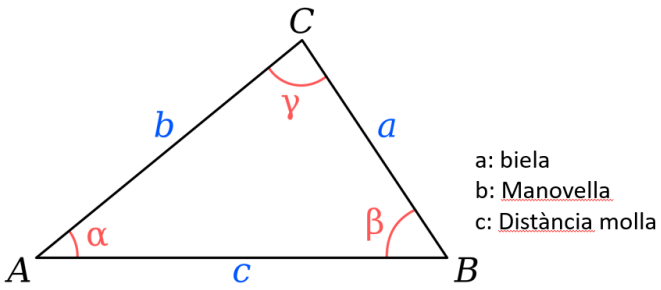


Figura 10 *Representació del teorema del cosinus. Extret de "Triangle with notations 2", Dweisman. Llicència CC.*

Seguidament, es poden veure les equacions que defineixen el teorema del cosinus:

$$a^2 = b^2 + c^2 - 2bc\ \cos\alpha \tag{9}$$

$$b^2 = c^2 + a^2 - 2ca\ \cos\beta \tag{10}$$

$$c^2 = a^2 + b^2 - 2ab\ \cos\gamma \tag{11}$$

Tenint en compte que la mida de la manovella (b) i la de la biela (a) es coneixen perquè són constants en el temps i, que la distància de la molla (c), és a dir, la posició de la molla s'obté amb les equacions explicades en l'apartat anterior, es pot operar amb una de les equacions per tal de conèixer l'angle α, ja que aquest és l'angle que el motor ha de moure la manovella per anar a cada una de les posicions de la molla. A continuació es pot veure com s'aconsegueix l'angle α. Si s'aïlla el cosinus de α l'equació *(9)* queda de la següent forma.

$$\frac{-a^2 + b^2 + c^2}{2bc} = \cos\alpha \tag{12}$$

Fent l'invers del cosinus es pot trobar el valor de l'angle tal com es pot veure a continuació.

$$\alpha = \cos^{-1}\left(\frac{-a^2 + b^2 + c^2}{2bc}\right) \tag{13}$$

L'equació anterior permet conèixer l'angle en funció de la posició en què es trobi la massa del mecanisme que s'està simulant.

4. Desenvolupament del projecte

En aquest apartat s'explica com s'ha dut a terme les parts mecàniques, electròniques i de programació per tal de desenvolupar el projecte d'acord amb els requeriments de règim de treball del mecanisme.

4.1. Règim de treball en el projecte

Com que aquest projecte pretén ser un recurs pedagògic per fer més fàcil la comprensió del funcionament d'una molla, s'ajustarà el sistema per poder simular oscil·lacions amb masses i constants de molles les quals permetin comprendre com és el moviment. Per tant, no es busca poder simular una combinació de paràmetres que com a resultat l'oscil·lació sigui molt ràpida, ja que si va molt de pressa tampoc ajuda a entendre el funcionament perquè és difícil comprendre'l.

Les restriccions en la simulació del moviment venen donades principalment per la constant de la molla, el coeficient de viscositat de l'esmorteïdor, l'abast màxim d'oscil·lació, la massa de l'objecte que se simula i la posició inicial a la qual s'inicia el moviment. Així que s'ha decidit que l'usuari pugui determinar la posició inicial del mecanisme, la massa que es vol simular, la constant d'esmorteïment i la constant elàstica de la molla.

La posició inicial del moviment serà un valor el qual s'haurà de trobar dins l'abast mecànic que permet el mecanisme, en canvi, la constant d'esmorteïment podrà ser qualsevol valor. La constant de la molla i la massa són uns paràmetres els quals es pot posar el valor que es vulgui, però condiciona directament amb la freqüència d'oscil·lació. Al mateix temps, té un efecte directe amb l'actuador encarregat de moure el mecanisme, ja que si la freqüència demanada és superior a la que pot oscil·lar l'actuador, aquest no permetrà representar una simulació ben feta.

4.2. Simulació parell motor

Abans d'escollir el tipus d'actuador necessari per moure el mecanisme, és necessari fer una sèrie de càlculs o simulacions per tal de conèixer els paràmetres claus del mecanisme. Aquest és el cas del parell motor, un paràmetre molt important en el moment d'escollir un tipus d'actuador o un altre. Per determinar el parell motor necessari per moure el sistema, s'ha fet una simulació del moviment amb el PTC Creo i s'ha realitzat un estudi dinàmic just en l'eix on l'actuador accionarà el mecanisme, aconseguint d'aquesta manera determinar el parell en cada posició possible.

Per fer-ho, s'ha fet un disseny simplificat del mecanisme només dissenyant la manovella, la biela, el patí i una guia lineal per on llisca el patí. A continuació es pot veure una captura de l'assemblatge del conjunt de peces citades anteriorment per fer la simulació, d'acord amb les dimensions del mecanisme.

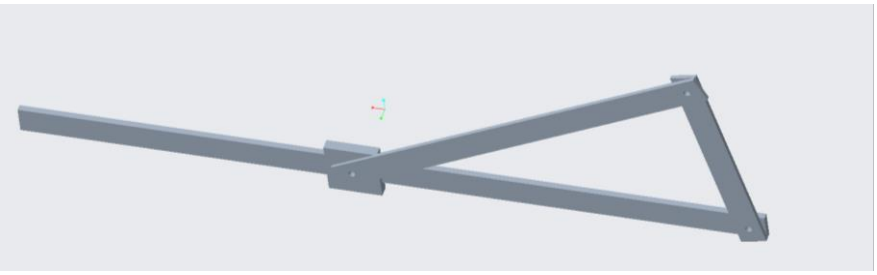
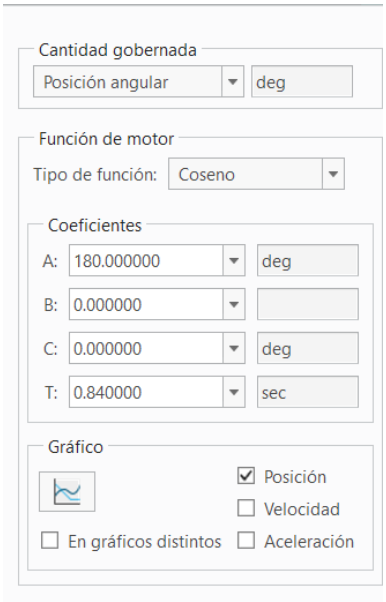


Figura 11 *Disseny mecanisme PTC Creo*

Una vegada muntat l'assemblatge anterior, s'ha efectuat l'anàlisi dinàmica per determinar el parell motor necessari. Primer de tot, és molt important que les peces siguin amb les unitats desitjades i del material correcte, en aquest cas alumini.

Per iniciar el procés de l'anàlisi, el primer que cal és activar l'efecte de la gravetat, en la direcció i sentit correcte. Una vegada fet, cal definir el motor encarregat de moure el mecanisme. Per fer-ho cal crear un motor cinemàtic que, en el cas del mecanisme d'oscil·lació de la molla, es mogui de forma sinusoidal. És per això que s'ha definit un motor en el qual es pugui controlar la posició angular seguint una funció de cosinus. A continuació es pot veure la configuració que se li ha assignat.



Es pot apreciar que s'ha configurat amb una amplitud d'oscil·lació de 180º, ja que amb aquests 180º completaria una oscil·lació. També s'ha establert un període de 0.84s, ja que d'acord amb la velocitat màxima del servomotor donada pel fabricant, aquest seria el temps mínim que podria tardar a fer un període.

Figura 12 *Configuració del motor*

Una vegada definides les característiques del motor i la posició del motor cal definir la mesura. Com que es vol mesurar el parell motor, cal que la mesura sigui de tipus càrrega neta i en l'eix on hi haurà el motor. A continuació es pot veure una captura d'aquesta configuració.

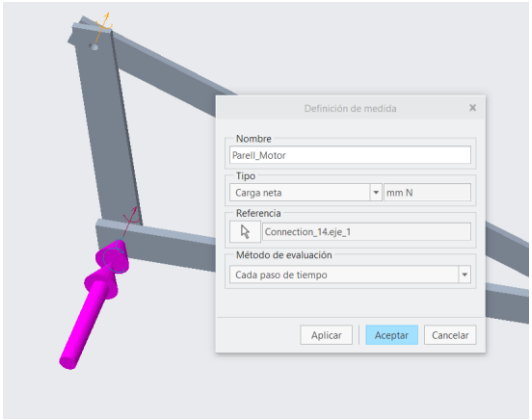


Figura 13 *Posicionament de la mesura del parell motor*

En el moment de definir l'anàlisi s'ha de tenir en compte configurar-lo com a dinàmic, ja que per determinar el parell motor s'ha de fer un estudi d'aquest tipus. En l'apartat de motors, cal verificar que detecta correctament el motor que s'ha creat en el pas anterior i, l'últim aspecte que cal tenir en compte, és habilitar la pestanya d'activar la gravetat i activar la fricció en l'apartat de les càrregues externes.

En les següents captures es pot veure la configuració de l'anàlisi.

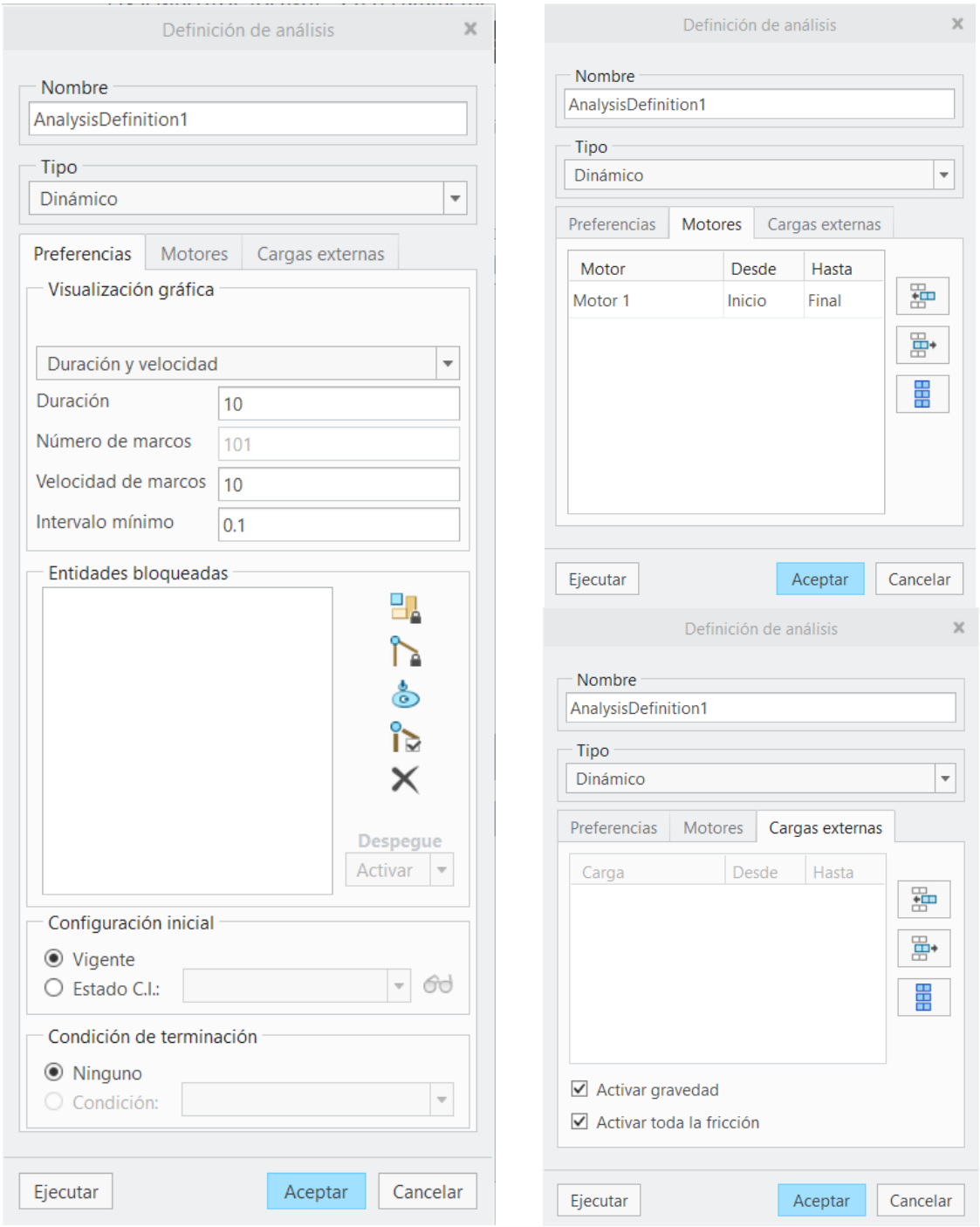


Figura 14 *Configuracions anàlisis dinàmic*

Per obtenir el resultat cal anar a mesures. La Figura 15 mostra la gràfica del parell motor en funció de la posició.

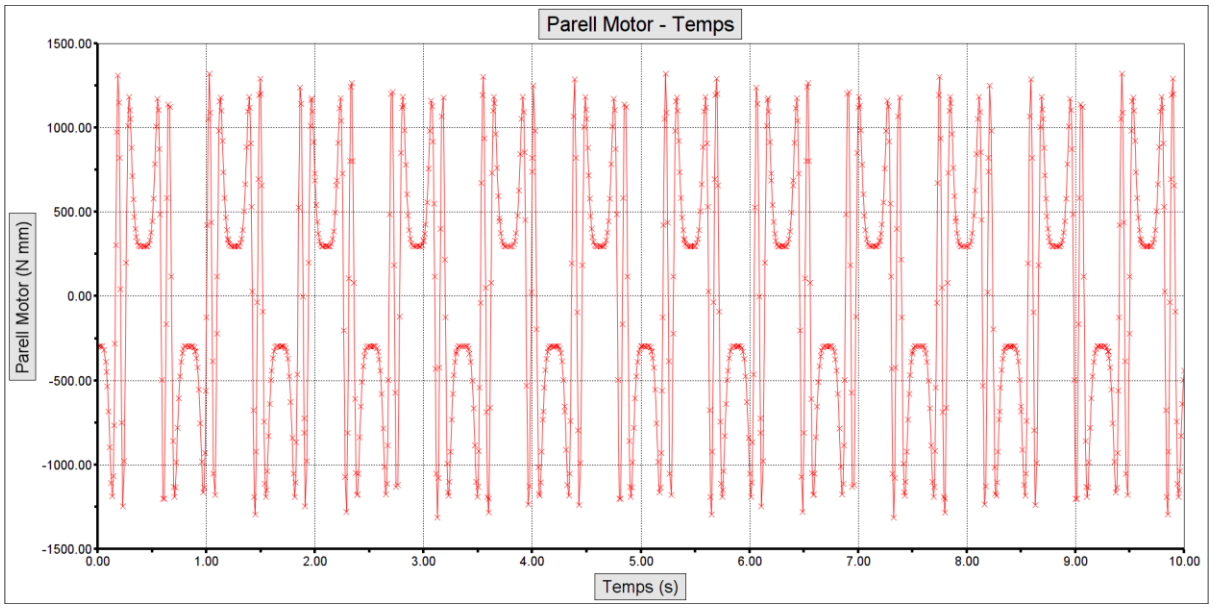


Figura 15 *Gràfic parell motor necessari*

Es pot apreciar en la gràfica obtinguda que el parell màxim és d'aproximadament 1.3N·m. Cal tenir en compte que s'haurà d'aplicar un coeficient de seguretat per tal d'assegurar el correcte funcionament del sistema.

4.3. Electrònica

La part d'electrònica del projecte està formada per tots els components electrònics que s'han utilitzat en el mecanisme. Principalment, de components electrònics hi ha l'actuador i la placa de programació.

4.3.1. Arduino Due

La placa de programació que s'ha fet servir és l'Arduino Due, ja que es requeria una velocitat de càlcul, de transmissió i lectura de dades pel port sèrie bastant gran, sobretot quan es tracta de donar les ordres de moviment des del Simulink. A més, el tutor del projecte va aconsellar l'ús d'aquesta placa i no la Uno perquè no dona tants problemes per sincronitzar-la amb el Simulink, ja que es va intentar amb la placa Arduino Uno i l'intent no va ser satisfactori, perquè no es va aconseguir el moviment correcte. A continuació es pot veure una imatge de la placa emprada.

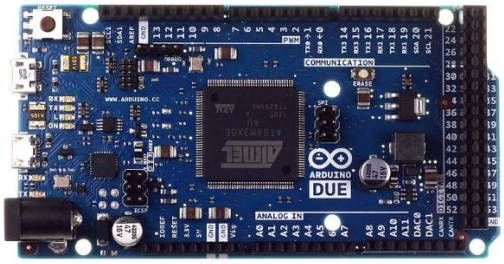


Figura 16 *Arduino Due. Extret de "Arduino Due Front", de Arduino SA. Llicència CC*

4.3.2. Actuador

Escollir l'actuador és una part molt important del projecte, ja que en funció de quin s'escull condiciona en l'apartat de disseny del mecanisme i de la programació. És per això que es va tardar força a escollir l'actuador que s'utilitzaria. Finalment, l'actuador usat per moure el mecanisme és un servomotor. Es va escollir un servomotor, perquè per completar una oscil·lació del mecanisme es requereix una rotació del motor de 180º i s'ha de controlar en cada instant en quina posició es troba l'actuador. Aquestes dues tasques són relativament fàcils d'aconseguir i controlar amb un servomotor. A part, les equacions que controlen el moviment, ens donen informació de la posició a la qual s'ha de trobar el sistema en cada instant de temps, per la qual cosa és

relativament senzill de transformar aquesta posició lineal a posició angular, tal com s'ha explicat anteriorment en la modelització matemàtica.

Així que com a requisits, cal que el parell del servomotor sigui bastant alt per moure el sistema i accelerar el sistema com cal en cada una de les posicions possibles. Addicionalment, un requisit molt important és que tingui una velocitat considerablement ràpida per tal de poder simular diverses combinacions de condicions.

La simulació dinàmica explicada prèviament va permetre comprovar que un servomotor era un actuador adequat, ja que no era una tasca molt complexa trobar un servomotor que complís els requisits de parell motor. En les especificacions dels servomotors, majoritàriament, el parell que té no és definit per unitats del sistema internacional sinó en unitats de massa (generalment kg) per unitats de distància (generalment cm). Així que cal aplicar un factor de conversió d'unitats per obtenir el valor amb unitats del sistema internacional.

Una característica crucial pel funcionament del projecte és la velocitat màxima a la qual es pot moure el servomotor. Tenint en compte el parell necessari obtingut en la simulació i que es vol fer oscil·lacions de fins a 1Hz de freqüència, el servomotor escollit es pot veure a la Figura 17.

Figura 17 *Servomotor 20kg. Extret de "ANNIMOS Servo Digital", ANNIMOS. Copyright.*

Es tracta d'un servomotor de 20kg·cm que té un rang de tensió d'alimentació d'entre 4.8V i 6.8V. El parell i la velocitat màxima subministrada depenen d'aquest valor de tensió. A continuació es pot veure una captura d'algunes de les especificacions importants del servomotor.

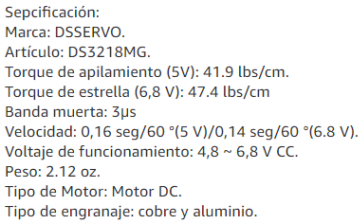


Figura 18 *Especificacions servomotor*

Per complir l'objectiu de poder fer simulacions del voltant de 1Hz, és necessari que el període més ràpid que pugui fer el servo sigui d'1 segon. És per això que s'ha calculat quin és el temps de període mínim que pot tardar en funció de la velocitat màxima a la qual es pot moure. La velocitat màxima, segons les especificacions del fabricant, és de 0.14s/60º. Com que en aquest projecte es pretén fer ones de tipus sinusoidal amb el servomotor, s'ha transformat aquesta velocitat amb el temps mínim que pot tardar a fer un període quan ha de recórrer l'amplitud màxima. A continuació es pot veure aquesta transformació:

$$T = \frac{0.14}{60} \cdot 360 = 0.84s \qquad (14)$$

Principalment, el què s'ha fet és calcular quant tarda a fer 1º i a continuació s'ha multiplicat per 360º, ja que són els graus que ha de recórrer per completar una oscil·lació en la màxima amplitud. Com a resultat del càlcul, s'obté que el període mínim és de 0.84s, per la qual cosa és possible fer oscil·lacions de fins a aproximadament 1.2Hz.

Per a complir els requisits de parell motor obtinguts en la simulació de parell motor necessari, s'ha usat la següent informació trobada també en les especificacions del motor.

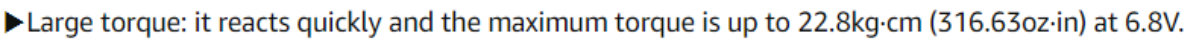


Figura 19 *Especificacions 2 servomotor*

En aplicar el canvi d'unitat s'obté un parell motor de 2.23N·m aproximadament, per la qual cosa té un factor de seguretat de 1.7 aproximadament respecte al necessari en la simulació.

Per a consultar l'esquema de connexions entre l'Arduino i el servomotor (vegeu Apèndix D Esquema de connexions).

4.4. Disseny 3D del mecanisme

Un disseny 3D en l'àmbit de la mecatrònica consisteix a realitzar un model tridimensional del prototip en el qual s'inclouen els components electrònics i mecànics que formaran el mecanisme.

Per fer el disseny s'ha utilitzat dos softwares diferents, el PTC Creo Parametric i el Solidworks. El motiu pel qual no s'ha usat un únic programari és perquè amb el software que s'està més familiaritzat com a usuari és el Solidworks, però amb aquest s'han tingut limitacions, ja que en tractar-se d'una versió molt bàsica i amb una llicència limitada, no es poden fer simulacions dinàmiques. Així que per fer les simulacions s'ha emprat el PTC Creo i pel disseny de les peces el Solidworks.

L'objectiu del disseny del mecanisme és aconseguir que sigui robust i manejable, així que un factor molt important és aconseguir que el pes de l'estructura no sigui molt elevat. Aquest factor ha portat a decidir que el material usat per la majoria de les parts del mecanisme sigui l'alumini, un material que pesa poc i amb propietats mecàniques que s'adeqüen a les necessitats del projecte. El fet de fer el sistema lleuger, afavoreix en el sentit de trobar un actuador que compleixi els requisits de parell necessaris per moure el sistema. A més, per disminuir al màxim els requeriments de parell necessari, s'ha decidit fer ús d'una guia del tipus Hiwin per representar el moviment lineal, ja que són reconegudes pel baix coeficient de fricció que tenen en lliscar. En referència al xassís del mecanisme, s'ha dissenyat emprant perfils d'alumini mentre que els suports, en la majoria dels casos, s'ha optat per dissenyar-los i imprimir-los amb la impressora 3D, un recurs ràpid, eficaç i barat.

En el procés de disseny, és important que les peces que es dissenyen i els components que s'inclouen a l'assemblatge siguin els més semblants possibles als components reals que després s'utilitzaran, ja que d'aquesta manera s'eviten problemes en el procés de construcció. És per això que s'ha fet ús del CAD subministrat pel proveïdor d'alguns productes. Aquest és el cas de la guia Hiwin, del patí Hiwin, del servomotor i de l'Arduino Due.

4.4.1 Peces dissenyades

A continuació es pot veure una taula de totes les peces les quals se n'ha obtingut un CAD. La majoria de les peces han estat dissenyades pròpiament, tot i que tal com s'ha citat anteriorment, peces comercials com és el cas de la guia Hiwin, el patí Hiwin, els perfils d'alumini, la placa Arduino Due i el servomotor s'ha descarregat i usat l'arxiu CAD proporcionat pel fabricant del producte o d'una pàgina web molt utilitzada en Solidworks anomenada Grabcad.

Taula 1 *Peces dissenyades*

Nom peça	Material peça	Obtenció	Obtenció Disseny	Referència plànol:
Manovella	Alumini	Manufacturat	Propi	Apèndix A.1 Plànol manovella
Biela	Alumini	Manufacturat	Propi	Apèndix A.2 Plànol biela
Guia Hiwin	Varis	Comprat	Fabricant	Apèndix A.3 Plànol guia Hiwin
Suport guia Hiwin	Alumini	Manufacturat	Propi	Apèndix A.4 Plànol suport guia Hiwin
Patí Hiwin	Varis	Comprat	Fabricant	Apèndix A.5 Plànol patí Hiwin
Suport Patí Biela	PLA	Impressió 3D	Propi	Apèndix A.6 Plànol suport patí biela
Lateral esquerre	Perfils d'alumini	Manufacturat	Propi	Apèndix A.7 Plànol lateral esquerre
Lateral dret	Perfils d'alumini	Manufacturat	Propi	Apèndix A.8 Plànol lateral dret
Suport horitzontal	Perfils d'alumini	Manufacturat	Propi	Apèndix A.9 Plànol suport horitzontal
Suport servomotor vertical	Perfils d'alumini	Manufacturat	Propi	Apèndix A.10 Plànol suport servomotor vertical

Suport servomotor horitzontal	Perfils d'alumini	Manufacturat	Propi	Apèndix A.11 Plànol suport servomotor horitzontal
Suport servomotor	PLA	Impressió 3D	Propi	Apèndix A.12 Plànol suport servomotor
Suport Arduino	PLA	Impressió 3D	Propi	Apèndix A.13 Plànol suport Arduino
Suport connexions	PLA	Impressió 3D	Propi	Apèndix A.14 Plànol suport connexions
Peus protecció	Perfils d'alumini	Manufacturat	Propi	Apèndix A.15 Plànol peus protecció
Protecció	Vidre plàstic	Comprat/Manufacturat	Propi	Apèndix A.16 Plànol protecció
Gruix biela-manovella	PLA	Impressió 3D	Propi	Apèndix A.17 Plànol gruix biela-manovella
Femella perfil alumini	Alumini	Comprat	Propi	Apèndix A.18 Plànol femella perfil alumini
Escaire perfil alumini	Alumini	Comprat	Propi	Apèndix A.19 Plànol escaire perfil alumini
PCB Arduino	Varis	Comprat	Grabcad	Apèndix A.20 Plànol Arduino Due
Servomotor	Varis	Comprat	Grabcad	Apèndix A.21 Plànol Servomotor

4.4.2. Assemblatge

Una vegada dissenyades les peces principals, es va procedir a avançar amb l'assemblatge, la qual cosa va permetre donar-se compte de possibles errors i de comprovar que el disseny compleix els requisits desitjats. Fer l'assemblatge també va permetre calcular distàncies i, per tant, ajustar les peces de forma que l'encaix entre elles sigui el desitjat. A continuació es poden veure algunes captures de l'assemblatge complet del mecanisme.

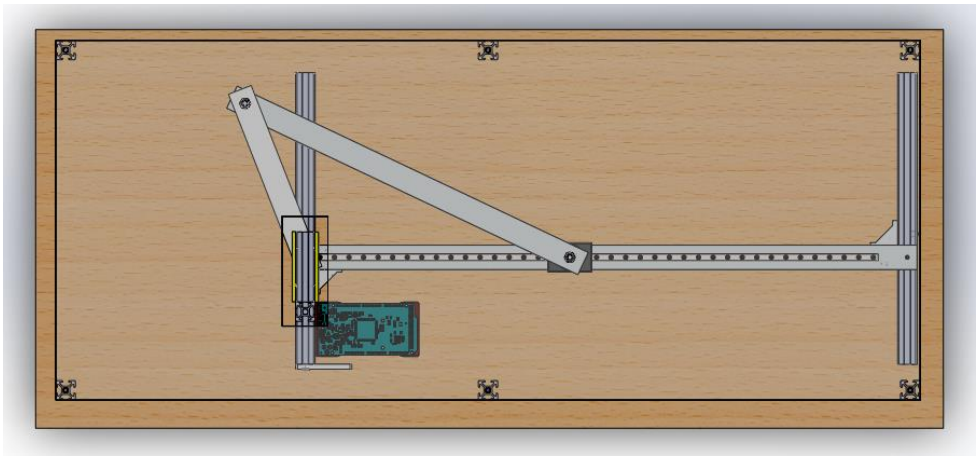


Figura 20 *Assemblatge final 1*

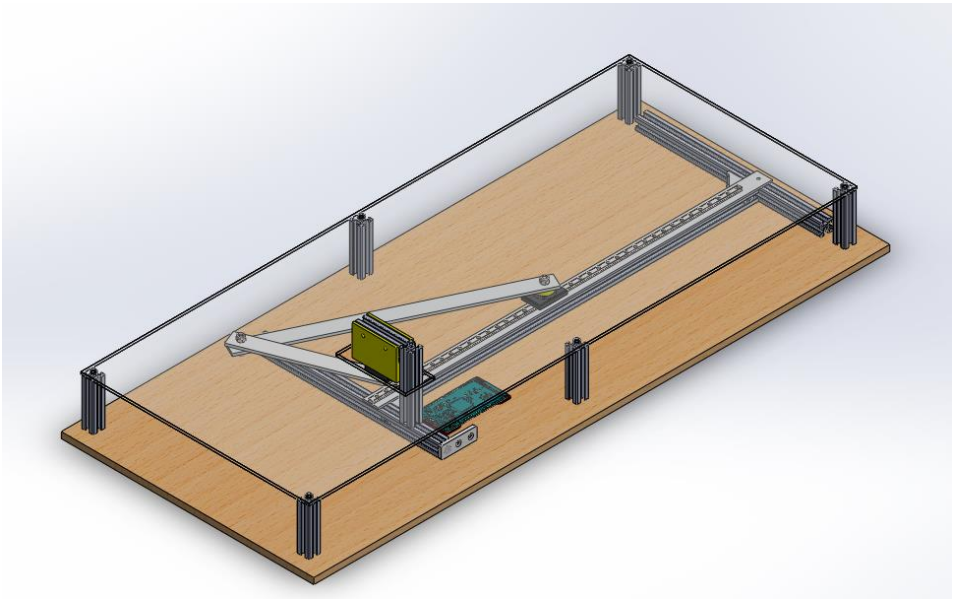


Figura 21 *Assemblatge final 2*

A la Figura 22 que es pot veure a continuació, hi ha mostrat l'especejament de l'assemblatge amb el nom de totes les parts que el formen.

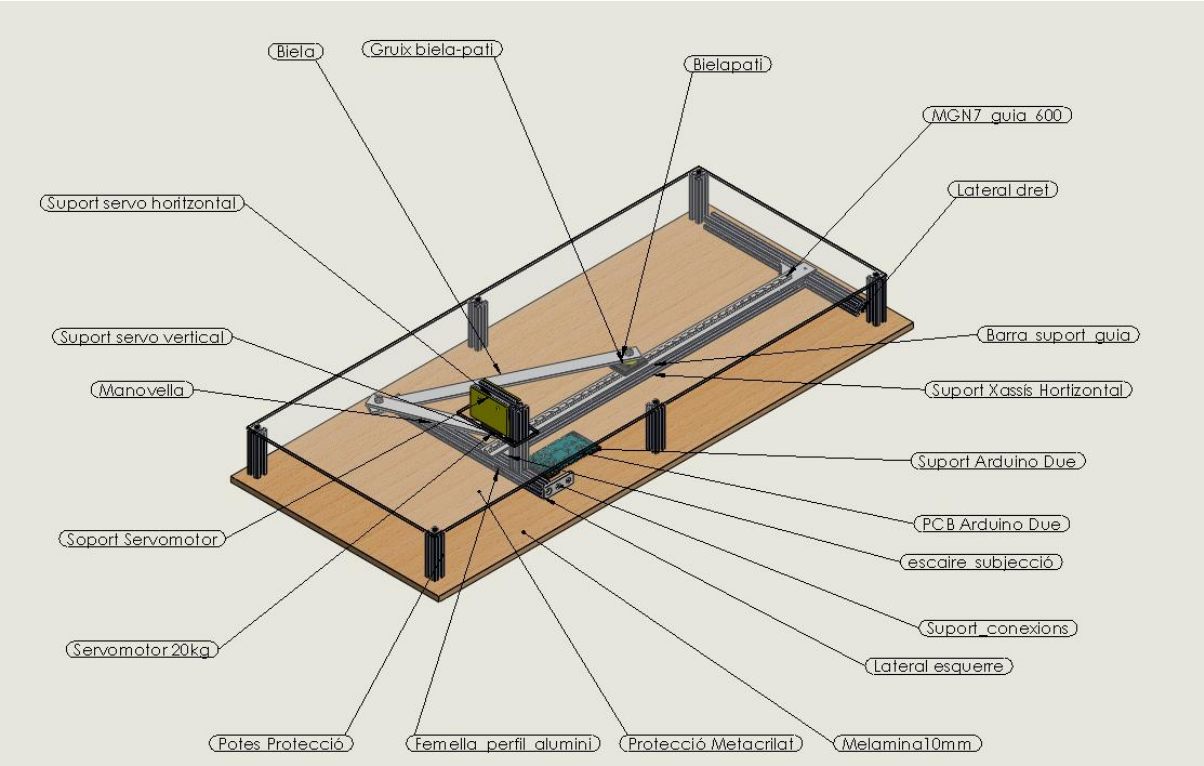


Figura 22 *Especejament assemblatge*

4.5. Programació

Per la programació del projecte, des de bon principi s'havia pensat programar els moviments de l'actuador amb el Matlab, però a mesura que es va anar avançant amb el projecte, es va creure que seria interessant que el sistema fos accessible per a tothom, és a dir, que no es requereixi programes amb llicències de pagament. Així que es va decidir també implementar la programació utilitzant l'Ardunio IDE. És per això que l'apartat de programació està dividit en Simulink i programació de l'Arduino IDE. A més, cal afegir que per la programació a través de l'Arduino IDE s'ha fet una aplicació per passar els paràmetres de simulació que es pretenen simular i veure en temps real una gràfica de l'oscil·lació.

4.5.1. Simulink

Simulink és una extensió de Matlab amb un entorn de programació amb blocs que s'usa per dissenyar i simular sistemes en diferents dominis. La forma en la qual es pretén treballar amb el Simulink és controlant els moviments del motor en funció dels resultats obtinguts de la funció de transferència o de les equacions que permeten controlar el moviment en el domini freqüencial. Així que el que es va fer és crear un sistema, organitzat per subsistemes encarregats cada un d'una funció específica. A continuació es pot veure una captura del sistema general.

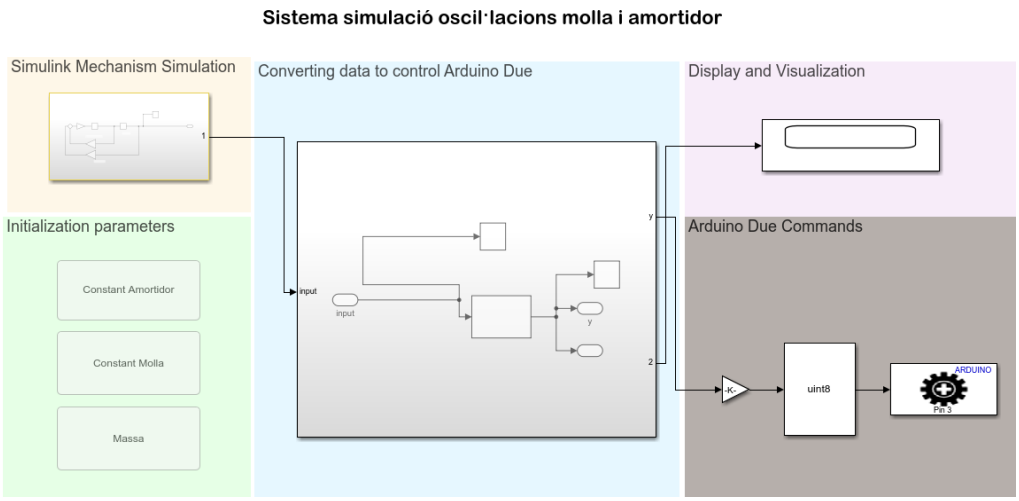


Figura 23 *Blocs programació Simulink*

Es pot apreciar que cada subsistema està envoltat d'una àrea d'un color diferent, conjuntament, amb un títol per distingir fàcilment cada un d'aquests subsistemes. A continuació s'explica quina funció té cada subsistema.

- Subsistema groc (*Simulink Mechanism Simulation*): en aquest subsistema es realitza tot el procés d'obtenir la posició del mecanisme per cada instant en funció dels paràmetres inicials introduïts per l'usuari.

  Així que en aquest subsistema hi ha implementat un sistema de blocs d'acord amb l'equació *(5)* per obtenir la posició. A continuació es pot veure una captura d'aquest sistema.

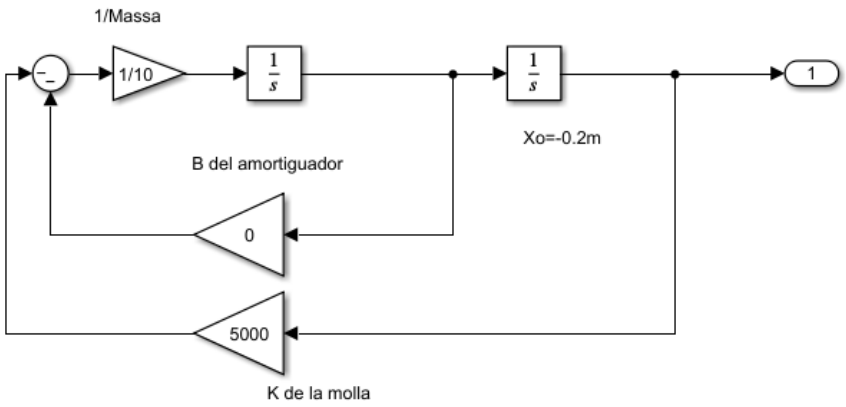


Figura 24 *Subsistema de càlcul del moviment amb Simulink*

- Subsistema verd (*Initialization parameters*): en aquest subsistema, l'usuari en prémer cada un dels polsadors li permet introduir el valor de la massa, de la constant de la molla i de la constant d'esmorteïment.
- Subsistema blau (*Converting data to control Arduino Due*): aquesta part és l'encarregada de transformar la posició obtinguda en el subsistema groc en un angle d'acord amb les equacions del teorema del cosinus explicades anteriorment, concretament a l'equació (*13*). A continuació es pot veure l'estructura que té aquest subsistema.

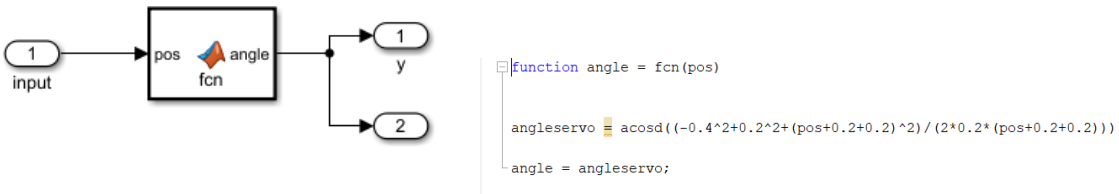


Figura 25 *Bloc de funció d'obtenció de l'angle*

- Subsistema violeta (*Display and Visualization*): mitjançant una funció del Simulink permet visualitzar l'angle en cada instant de temps.

- Subsistema gris *(Arduino Due Commands*): en aquest apartat el que es fa és escalar el valor de l'angle per un servo de 270º, es converteixen les dades a *uint8*, ja que és el format que es requereix per comunicar-se amb l'Arduino utilitzat i, finalment, l'últim bloc se li configura el pin de l'Arduino on hi haurà connectat el servomotor.

Durant el procés de proves, una vegada implementat el model amb el Simulink, es va tenir un problema el qual es va tenir força dificultat per superar-lo. Com que per controlar un servomotor com el que s'usa en el mecanisme s'utilitza la modulació per amplada de polsos (PWM), hi havia moments en el qual el motor no es movia i el mecanisme quedava parat, ja que el motor es trobava en una zona de banda morta o *dead zone*. Per comprovar i descobrir que era aquest l'error que es tenia quan el servomotor es quedava parat, es va procedir a analitzar el senyal que enviava l'Arduino al servomotor amb l'oscil·loscopi. Va ser en aquest punt que es va veure clar que l'error procedia d'aquesta zona morta.

Hi ha una sèrie de formes per reduir o escollir fer una acció o una altra quan el període coincideix amb aquesta zona, però com que encara no havia arribat l'ordre d'on s'havia de posicionar el servomotor en la següent iteració, no funcionaven aquestes possibles solucions. Per resoldre aquest problema, es va procedir programant des del Simulink, però aquesta vegada es va utilitzar un bloc de programació especialitzat en servomotors. Aquest, permet trametre constantment el valor de l'angle al qual s'ha de moure el servomotor. D'aquesta manera, com que constantment s'envien ordres d'on s'ha de posicionar el motor, fa que no s'hagi de tenir en compte aquesta banda morta. A continuació es pot veure una captura del bloc de programació de PWM emprat en el Simulink.

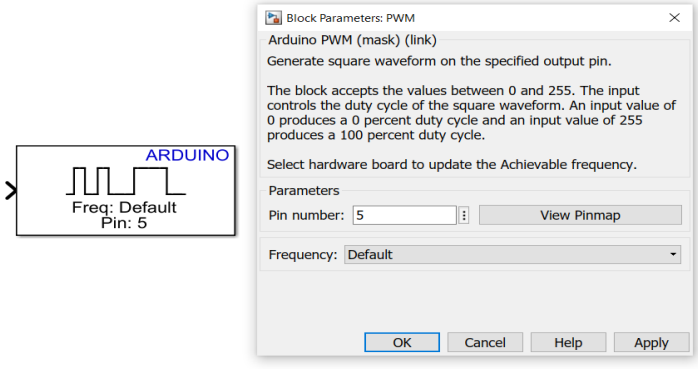


Figura 26 *Bloc Simulink comunicació Pin Arduino PWM*

Per programar aquest bloc només cal indicar el Pin on estarà connectat el servomotor a l'Arduino i si es vol, es pot establir una freqüència concreta o deixar-la per defecte. Aquest bloc genera una ona quadrada en funció del valor que li arriba a l'entrada. Aquest valor d'entrada ha de trobar-se entre 0 i 255 i en funció d'aquest el que es fa és ajustar el *duty cycle*, és a dir, el temps que es troba a nivell alt una ona quadrada durant un període. Però, tal com s'ha explicat anteriorment, en aquest bloc hi havia el problema de la banda morta. El bloc finalment utilitzat per solucionar el problema citat anteriorment es pot veure en la següent imatge:

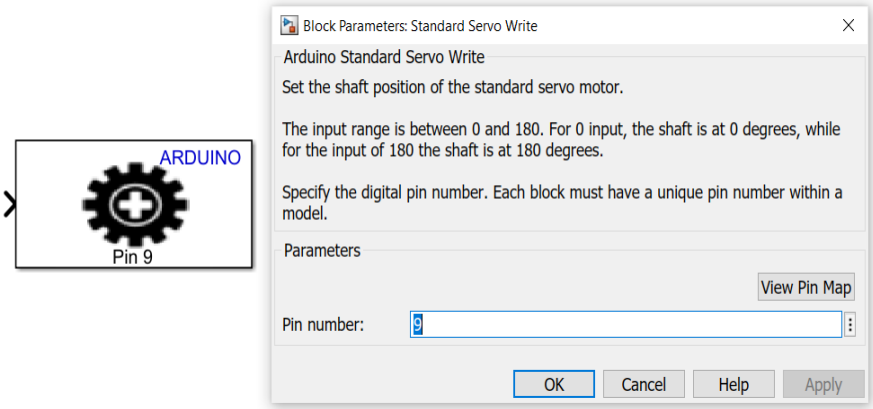


Figura 27 *Bloc Simulink comunicació Pin Arduino per servomotor*

Aquest bloc, a diferència del de PWM, s'entren valors entre 0 i 180. Aquests valors es veuen reflectits en l'angle el qual es mourà el servomotor. Si s'utilitza un servomotor de 180º, en el moment en què aquest bloc rebi un 0, enviarà al servomotor connectat que es desplaci fins a la seva posició de 0º. En canvi, si es tramet qualsevol altre valor, es posicionarà a la posició indicada en el valor d'entrada del bloc. Es tracta d'un bloc el qual té un funcionament impecable per un servomotor que es mogui 180º, però en el cas d'usar un servomotor que no tingui aquest abast de recorregut, el què cal fer és escalar el valor d'entrada, ja que si a l'entrada d'aquest bloc hi arriba un 180, el que farà és que el servomotor que hi hagi connectat es posicioni al seu angle màxim de desplaçament que pot assolir. Així que com que el servomotor emprat és de 270º ha calgut escalar els valors.

4.5.2. Programació amb Arduino IDE

El motiu pel qual es va decidir fer un codi de programació totalment independent del Simulink és per evitar que sigui necessari tenir llicència del Matlab per fer funcionar el mecanisme i, per tant, fer que el sistema sigui accessible per a tothom. A part, és una mica complicat el procés de detectar, localitzar i configurar la placa de programació en l'entorn de Simulink. A més, si se li suma que és bastant lent el procés de compilar des del Matlab i establir la connexió amb l'Arduino, fa que simular amb aquest entorn no sigui pràctic ni agradable. És per això, doncs, que programant mitjançant l'Arduino IDE es pretén fer més fàcil, ràpida i agradable la simulació.

Per fer el codi de programació, primerament, es va pensar com seria la programació i es va fer un ordinograma per tal de tenir en compte el màxim d'imprevistos en el moment de programar. Gràcies a l'elaboració d'un ordinograma o diagrama de flux, va permetre fer un esquema per estructurar el codi i tenir clar com s'ha de dur a terme abans d'iniciar el procés de programació.

En el cas de la programació amb l'Arduino IDE, el codi s'ha estructurat amb la seqüència principal en la qual es duen a terme les accions principals i dues subrutines que permeten fer càlculs i operacions d'ajustament deixant d'aquesta manera la seqüència principal una mica més neta. Principalment, el què es fa és inicialitzar les variables i a continuació esperar a rebre dades pel port sèrie. Aquestes dades són la massa, la constant de la molla, la constant d'esmorteïment i la posició inicial del mecanisme. Una vegada assignades aquestes dades a una variable, es comprova mitjançant una subrutina que l'actuador pugui complir els requisits de freqüència d'oscil·lació demanats en funció dels paràmetres escollits per l'usuari. Tot seguit, mentre una variable declarada anteriorment com a *true* es mantingui a *true,* el codi segueix amb el bucle de la seqüència, el qual consisteix a comprovar si hi ha més dades per llegir o no. En el cas que hi hagi dades pendents de llegir, es llegeixen i en funció de quina dada s'hagi rebut es realitza una acció o una altra. Les dades que es poden rebre són les següents:

- * : Si es rep aquesta dada significa que s'ha de parar la simulació i, per tant, es posen les variables anomenades *start* i *i* com a *false*.
- /: Si es rep aquesta dada s'estableix el temps a zero i segueix la seqüència.

En el cas que no hi hagi dades a llegir es procedeix amb la seqüència comprovant si la variable *start* es troba en estat *true* o *false*. Si es troba en estat *false,* torna a l'inici de l'operació, en canvi, si està en estat *true,* segueix la seqüència guardant el temps actual i fent els càlculs del temps de mostreig, acceleració, velocitat, posició, període i freqüència. A continuació, es procedeix a enviar per port sèrie un caràcter per saber quines dades s'envien i seguidament, s'envia el temps de mostreig, la posició i la freqüència. Per acabar, es crida una subrutina que calcula l'angle que s'ha de posicionar l'actuador i s'indica que el motor es posicioni a l'angle calculat. En la Figura 28 es mostra l'ordinograma de la seqüència principal:

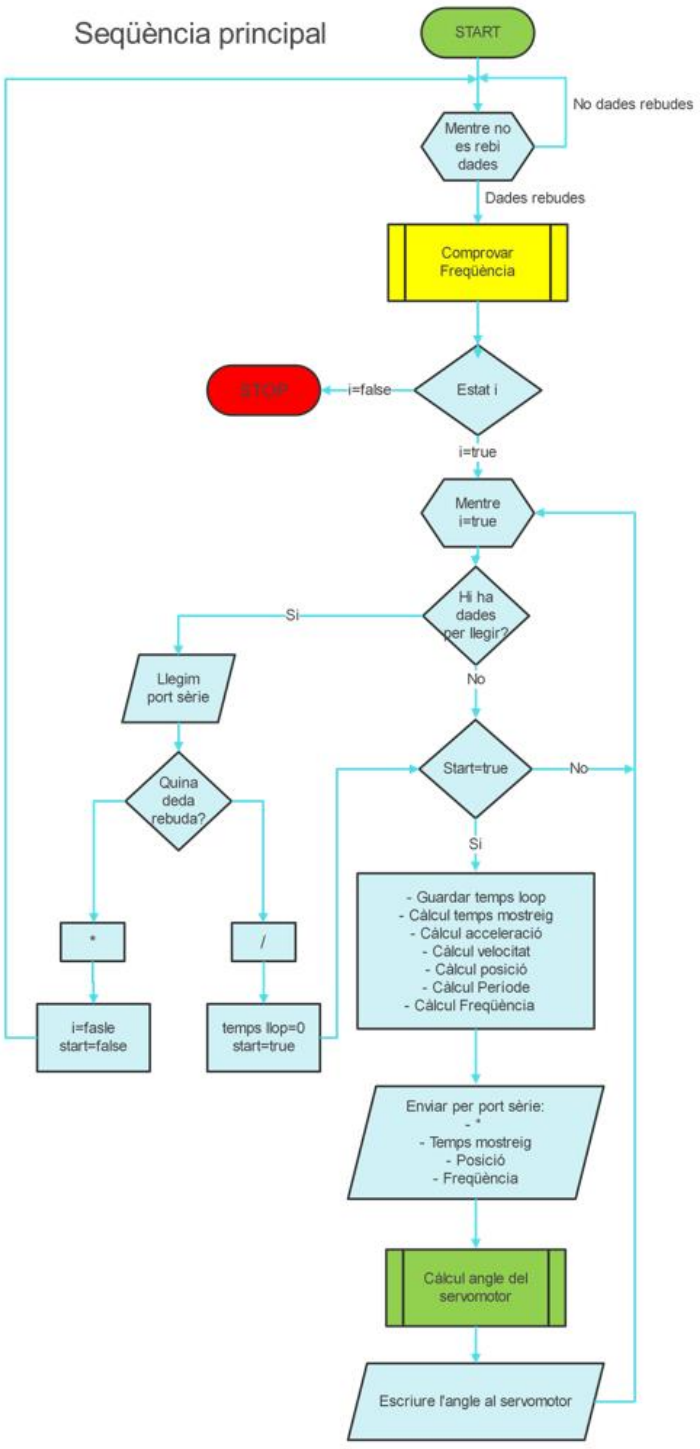


Figura 28 *Ordinograma seqüència principal*

Es pot apreciar que hi ha un bloc de color verd i un altre de color groc referent a les subrutines citades anteriorment. A continuació es pot veure l'ordinograma d'aquestes subrutines:

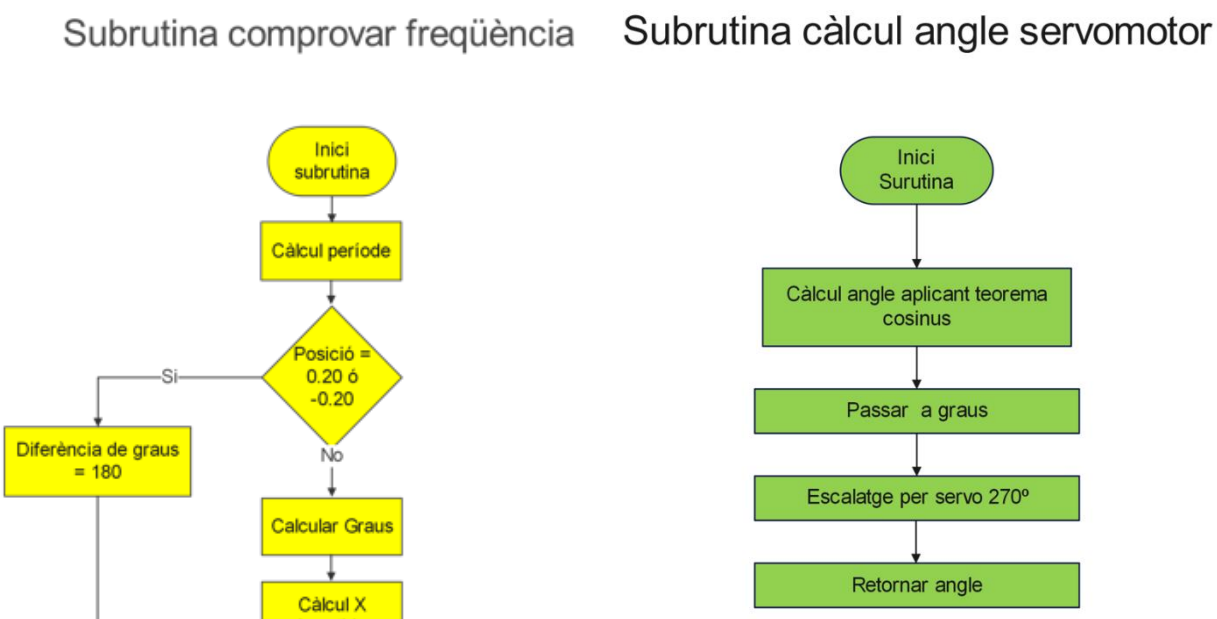


Figura 29 *Ordinograma càlcul angle*

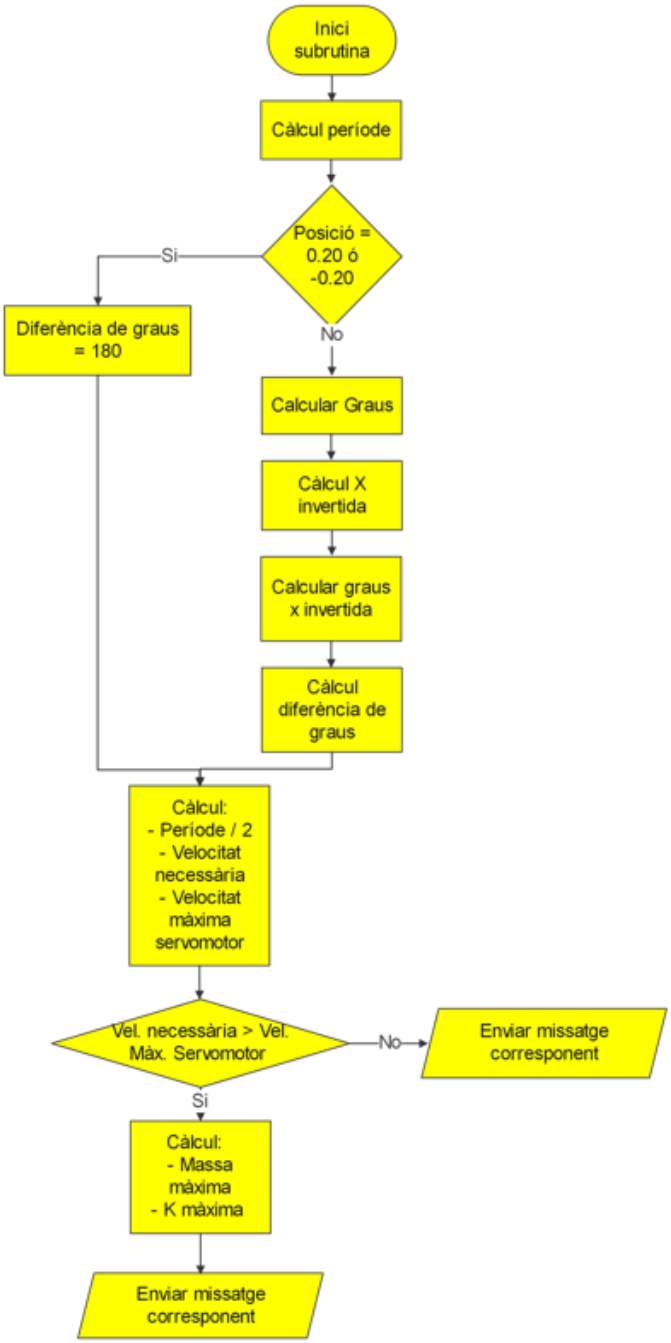


Figura 30 *Ordinograma comprovar freqüència*

Tal com mostren els ordinogrames anteriors, el groc correspon a la subrutina de comprovar que la freqüència sigui vàlida, és a dir, que la velocitat del servomotor sigui suficient ràpida i, el verd, és l'encarregat de calcular l'angle.

Començant per la subrutina de comprovar la freqüència, el primer que es realitza és calcular el període i es comprova si la posició inicial és 0.2 o -0.2. En el cas de no ser així, es calcula els graus de la posició a la qual s'inicia la simulació i, seguidament, s'inverteix la posició inicial per calcular els graus a l'altre extrem de l'ondulació. Després cal trobar la diferència de graus en valor absolut. En el cas que la posició inicial sigui 0.2 o -0.2, no cal fer els càlculs anteriors sinó que la diferència de graus és 180º, ja que el mecanisme, per anar d'un extrem a l'altre, ha de moure's 180º. Seguidament, cal fer una sèrie de càlculs per obtenir el valor del període, la meitat del període, la velocitat necessària per complir l'oscil·lació i la velocitat màxima de l'actuador, la qual és definida pel fabricant. Si la velocitat necessària és més gran que la velocitat màxima del motor, es calcula la massa i la constant de la molla per tal de poder fer la simulació i s'envia un missatge que es mostrarà a l'aplicació indicant que la velocitat supera la màxima i es mostren els valors límits de la massa i la constant de la molla calculats per poder fer la simulació. En el cas de ser suficient la velocitat del motor, només es tramet un missatge indicant que les dades entrades es poden simular.

Per la subrutina de calcular l'angle del servomotor, s'utilitza el teorema del cosinus explicat anteriorment a l'equació *(13),* es passa l'angle a graus i després s'escala el valor de l'angle per un servo de 270º, ja que la llibreria utilitzada permet controlar un servomotor de 180º dient-li l'angle que vol que es posicioni. En tractar-se d'un servomotor de 270º, si s'enviés que es mogui a la posició de 180º, entendria que s'ha de moure a la seva màxima amplitud, és a dir, 270º. És per això que es fa l'escalatge. Finalment, aquesta subrutina retorna el valor de l'angle per després donar-li l'ordre al servomotor de posicionar-se a l'angle corresponent.

4.5.3. Programació de l'aplicació

En aquest apartat s'explica com s'ha creat una aplicació per fer més fàcil la tasca d'entrar dades i visualitzar en temps real com està evolucionant la simulació. D'aquesta manera la comunicació entre l'usuari i el mecanisme es fa molt més agradable, senzilla i visual.

Per fer l'aplicació s'ha utilitzat una extensió de Matlab que s'anomena *App Designer*. L*'App Designer* permet programar aplicacions creant una interfície de forma bastant ràpida i senzilla. Els passos que s'han seguit per fer l'aplicació són els recomanats per Matlab. Aquests es poden veure detallats a continuació.

1. **Disseny de la interfície d'usuari**

   En aquest apartat, si es vol, no és necessari picar codi de programació sinó que es poden arrossegar objectes ja creats per les llibreries del Matlab i posicionar-los al lloc on calgui, com seria el cas de, per exemple, posar un requadre d'entrada de text. Fent-ho d'aquesta manera es genera de forma automàtica un codi que indica que s'ha posat, per exemple, un polsador amb una forma i dimensions determinades a unes coordenades específiques de l'aplicació. D'aquesta manera, dissenyar la interfície és relativament ràpid i fàcil. Utilitzant aquestes llibreries és com s'ha creat la interfície d'usuari. A continuació es pot veure una taula que resumeix els objectes que s'ha usat i una petita explicació del perquè s'utilitza cada un.

Taula 2 *Camps/objectes de l'aplicació*

Objecte	Funcionalitat
Camp numèric editable (Massa)	Aquest camp permet a l'usuari entrar el valor de la massa en kg que es vol simular.
Camp numèric editable (B)	Aquest espai permet recollir el valor de la constant d'esmorteïment definida per l'usuari.
Camp numèric editable (K)	El camp de la constant elàstica de la molla (K), permet a l'usuari entrar el valor que es desitja d'aquesta constant per fer la simulació.
***Slider* posició inicial**	El *Slider* o camp lliscant permet introduir en quina posició es vol iniciar el mecanisme.
Camp numèric editable (Posició inicial)	La funcionalitat que té aquest camp principalment és mostrar el valor de forma digital que el *Slider* està en cada moment.
Polsador (*Send Data*)	Aquest polsador quan és premut crida una funció que indica que s'ha d'enviar les dades.
Polsador (*Start*)	En aquest cas, l'acció que es realitza una vegada es prem el polsador és enviar un caràcter per donar ordres d'iniciar la simulació.
Polsador (*Stop*)	A l'invers del polsador de *Start*, aquest envia un caràcter per parar la simulació.
Camp numèric (Freqüència)	Aquest camp mostra el valor de la freqüència que està treballant el mecanisme.
Camp numèric (Temps total)	En el requadre de temps total es mostra en temps real quant de temps ha passat des que s'ha iniciat la simulació.
Camp de text (Advertències)	Aquest camp de text permet mostrar missatges d'advertències que s'envien des de l'Arduino.
Gràfic	El gràfic permet dibuixar la posició a la qual es troba la massa que s'està simulant en cada instant de temps i en temps real.

D'acord amb els objectes explicats en la taula anterior, la Figura 31 mostra el resultat del disseny de la interfície utilitzada per a l'aplicació.

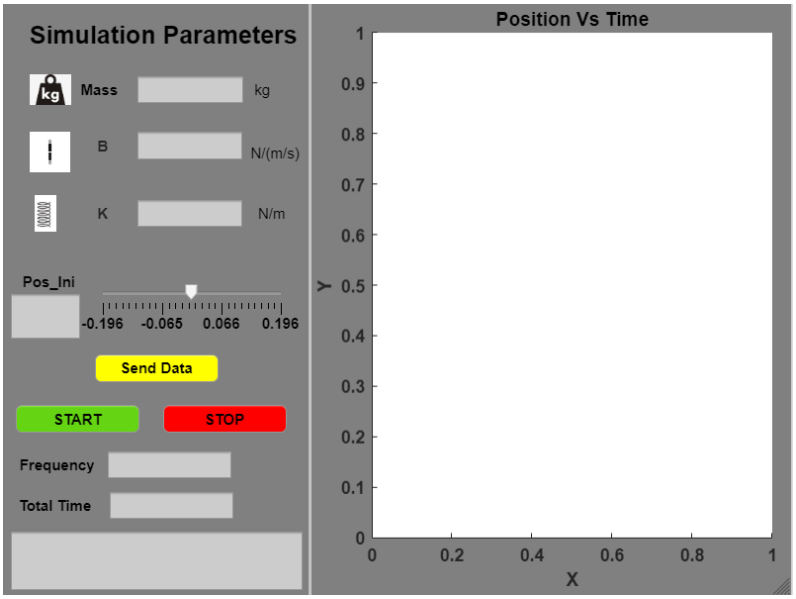


Figura 31 *Captura interfície aplicació*

A més dels camps explicats anteriorment a la Taula 2, en la Figura 31 es pot veure que també s'ha posat la correspondència de cada camp, les unitats del valor i una imatge petita d'una molla, una massa i un esmorteïdor per fer l'aplicació més visual. Es pot apreciar també que s'ha modificat el color dels polsadors, dels requadres i del fons.

2. Programació del comportament de l'aplicació

Aquesta part consisteix a escriure codi per establir la comunicació amb l'Arduino i per definir quina acció té associada cada un dels objectes que s'han explicat anteriorment. Per programar s'utilitza el llenguatge de Matlab tot i que també hi ha opció de programar amb altres llenguatges. La forma en la qual s'ha desenvolupat aquest codi de funcionament de l'aplicació es pot veure resumit amb els següents ordinogrames. La Figura 32 correspon a la seqüència principal.

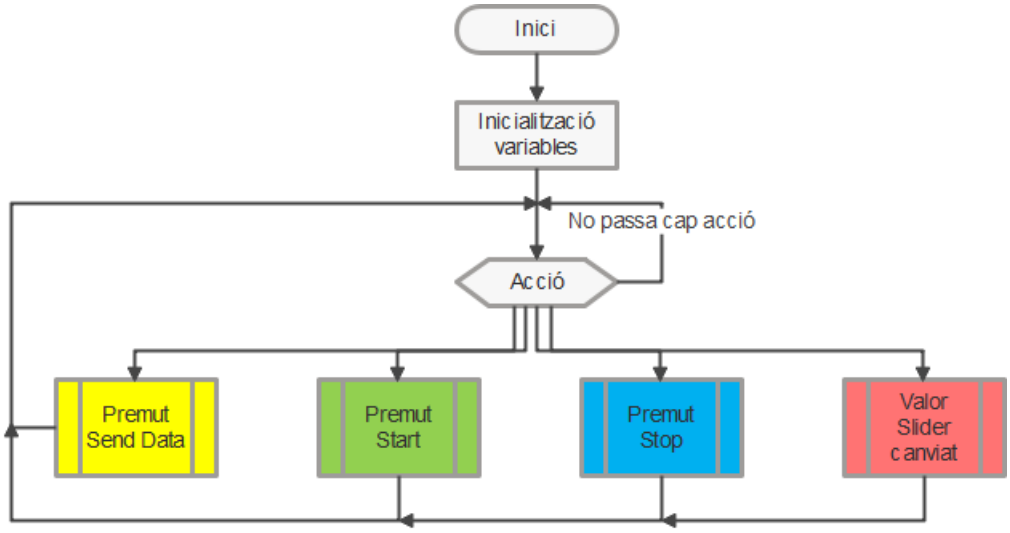
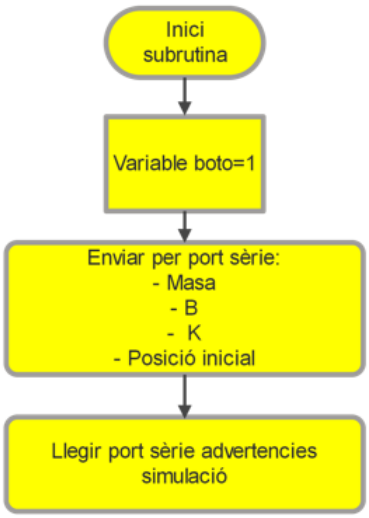


Figura 32 *Ordinograma principal aplicació*

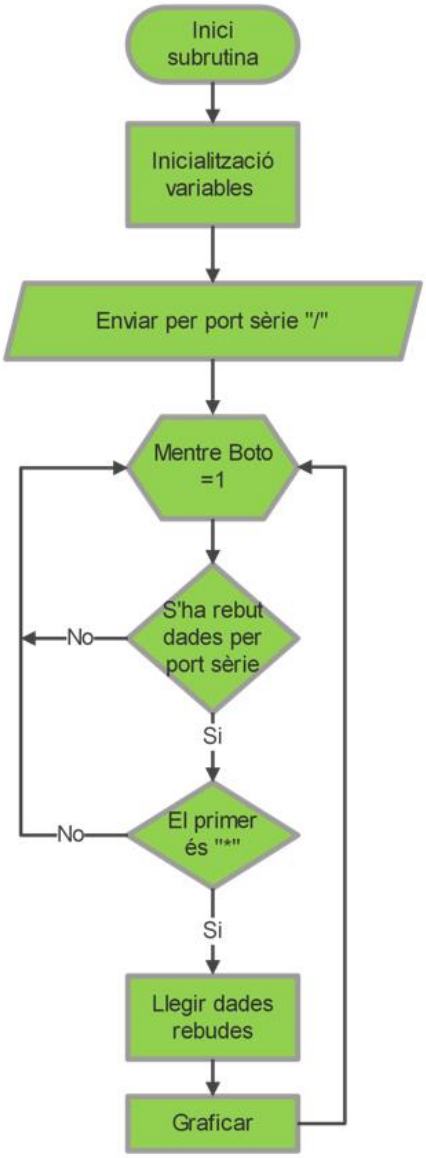
La seqüència s'inicia just en el moment de posar en marxa l'aplicació. El primer que es fa és inicialitzar les variables i esperar que passi alguna de les accions que estan definides com a subrutines. Aquestes accions poden ser prémer el botó de *Send data*, prémer *start, stop* o canviar el valor del *slider*. És a dir, aquest codi no avança fins que no es produeix una de les accions citades anteriorment les quals tenen una subrutina associada. La Figura 33 mostra l'ordinograma associat amb l'acció de prémer el botó *Send Data.*



En el moment que es polsa el botó d'enviar dades es posa a 1 una variable anomenada *botó.* A continuació, s'envia pel port sèrie el valor de la massa, la constant d'esmorteïment, la constant de la molla i la posició inicial entrades per l'usuari. Finalment, es llegeix i s'escriu al requadre de text d'advertències el missatge tramès des de l'Arduino pel port sèrie referent a si és possible o no fer la simulació.

Figura 33 *Ordinograma premut polsador enviar*

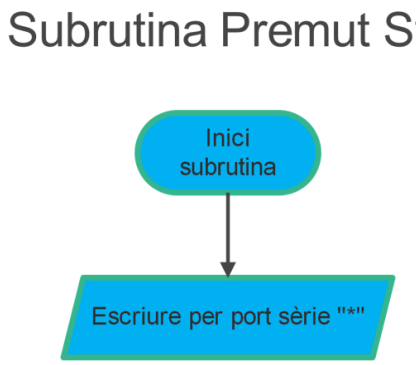
La següent subrutina és l'encarregada d'iniciar el procés de simulació en el moment en què es polsa aquest polsador. La Figura 34 permet veure l'ordinograma del funcionament.



En el moment que es polsa el botó de *Start,* es posen a zero les variables i s'inicialitzen les necessàries per aquesta subrutina. A continuació s'envia pel port sèrie un "/", per indicar al codi d'Arduino que s'iniciï la simulació. Tot seguit s'entra en un bucle que mentre la variable *boto* valgui 1, o més ben dit, mentre no es premi el botó *Stop*, es comprovi si s'ha rebut dades. Si no s'han rebut es torna a l'inici del bucle. En cas contrari, es comprova si el primer caràcter rebut és "*". En cas de no ser-ho es torna l'inici del bucle. Si es tracta del caràcter en qüestió, es llegeixen les dades de posició, temps de mostreig i freqüència, s'escriuen en el requadre corresponent i s'actualitza la gràfica de la posició en funció el temps.

Figura 34 *Ordinograma premut polsador Start*

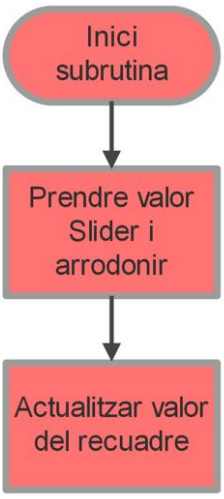
La subrutina encarregada d'indicar que es pari la simulació segueix un ordinograma molt simple que es pot veure a continuació.



En el moment en què es fa un clic al botó *stop* s'envia per port sèrie un "*", per indicar que s'ha de posar pausa a la simulació.

Figura 35 *Ordinograma premut polsador stop*

Per acabar, l'última subrutina és l'encarregada d'actualitzar el valor del requadre per mostrar el valor analògic del *slider* digitalment. A continuació es pot veure el diagrama de funcionament d'aquesta part.



Quan es canvia el valor del *slider* es llegeix el valor d'aquest, s'arrodoneix i es mostra en el requadre corresponent.

Figura 36 *Ordinograma actualitzar valor slider*

3. Crear l'executable

Aquest apartat permet crear un arxiu executable el qual es pot instal·lar a qualsevol ordinador amb sistema operatiu Windows. A continuació s'explica els passos que s'ha seguit per crear l'executable.

Primer de tot, dins l'*App designer* cal prémer *Share* i seleccionar l'opció *Standalone Desktop App* tal com es pot veure a la Figura 37.

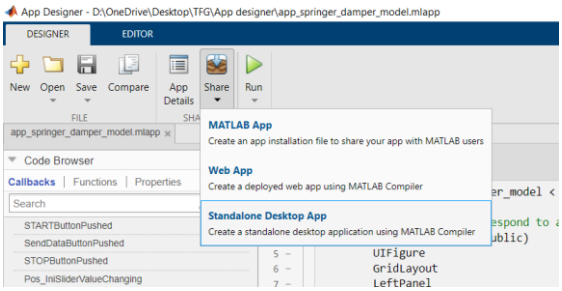


Figura 37 *Crear l'executable*

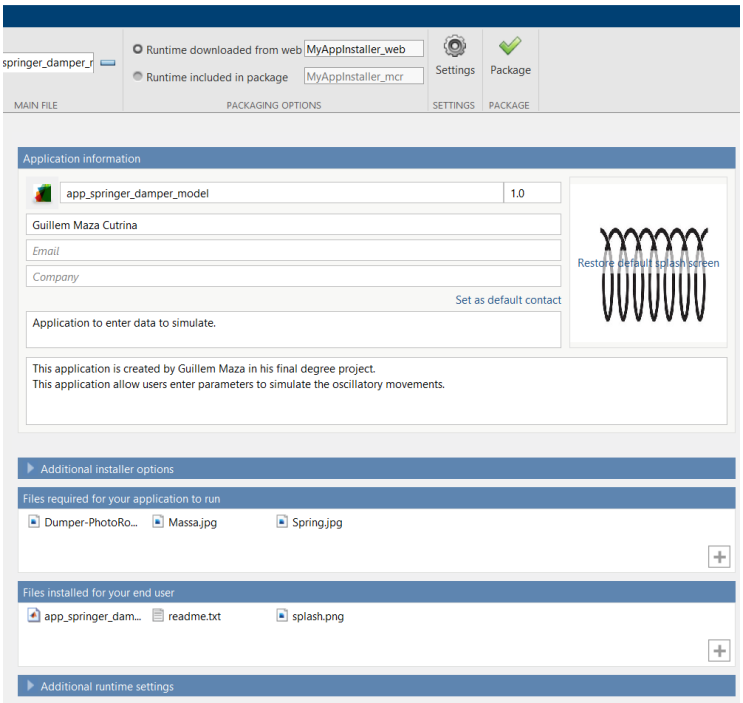


Figura 38 *Descripció de l'aplicació*

Seguidament, cal entrar les dades i prémer *Package* a la part superior tal com es pot apreciar a la Figura 38. Posteriorment, es crea un executable que només cal instal·lar com qualsevol altre programa d'ordinador.

4.6. Construcció del mecanisme

Per dur a terme la construcció del mecanisme, com que ja s'havia fet prèviament el disseny del mecanisme i, per tant, de cada una de les peces que el formen, es va fer ús dels plànols de les peces per tal de fabricar-les iguals que en el disseny. Com ja s'ha comentat, la major part del mecanisme està fet d'alumini o de peces que s'han obtingut per impressió 3D amb PLA. El material necessari per a la construcció es pot veure resumit en la Taula 3:

Taula 3 *Material necessari per a la construcció*

Component	Quantitat
Perfil alumini 20x20mm	1.9m
Guia Hiwin	1
Patí Hiwin	1
Placa Alumini 640x25x3mm	1
Biela	1
Manovella	1
Gruix biela-manovella	2
Metacrilat 1000x600x4mm	1
Suport biela patí	1
Femelles M4 perfil 20x20mm	10
Escaires perfil alumini 20x20mm	2
Cargols M4	10
Cargols M5	17
Femelles M5	5
Volanderes de goma M5	12
Suport Servomotor	2
Suport Arduino Due	1
Suport connexions	1
Placa melamina 900x420x10mm	1

El procés de construcció que s'ha seguit és, primerament, mecanitzar les peces per obtenir les dimensions requerides segons els plànols de disseny per, a posteriori, poder fer la construcció. La primera part que s'ha construït és el xassís del mecanisme, per a després, anar posant els components que permeten el moviment d'aquest. A continuació, s'ha procedit amb la instal·lació del servomotor amb el suport corresponent i, finalment, s'ha muntat els suports de la placa Arduino Due i el de l'alimentació. Treballar amb aquest ordre ha permès poder avançar amb la programació, fer proves i acabar de construir el mecanisme en paral·lel. La Figura 39 mostra el mecanisme fins a aquest punt de muntatge.

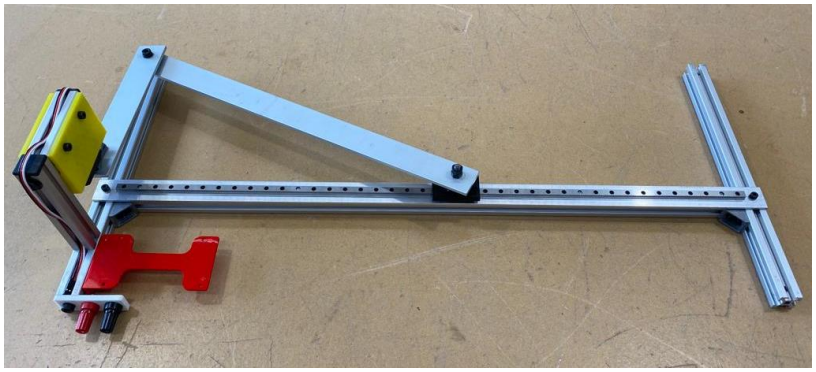


Figura 39 *Mecanisme sense protecció*

Una vegada es va tenir el mecanisme construït i testejat, va semblar que el sistema podia ser perillós en cas de posar les mans de forma involuntària mentre funciona. És per això, que es va decidir fer un sistema de protecció. Aquest, consta d'una placa de metacrilat, que permet veure perfectament a través d'ell i dificulta l'accés al mecanisme disminuint així les possibilitats de fer-se mal de forma involuntària. També disposa d'una placa de melamina en la qual va collat el mecanisme i les potes que suporten el metacrilat. Per tallar el metacrilat i la placa de melamina es va fer ús de les instal·lacions de l'Art de la Fusta Mecanitzats per fer els talls amb una seccionadora, una màquina ideal per talls rectes i nets.

A continuació es poden veure una sèrie d'imatges del mecanisme amb el sistema de protecció ja col·locat.

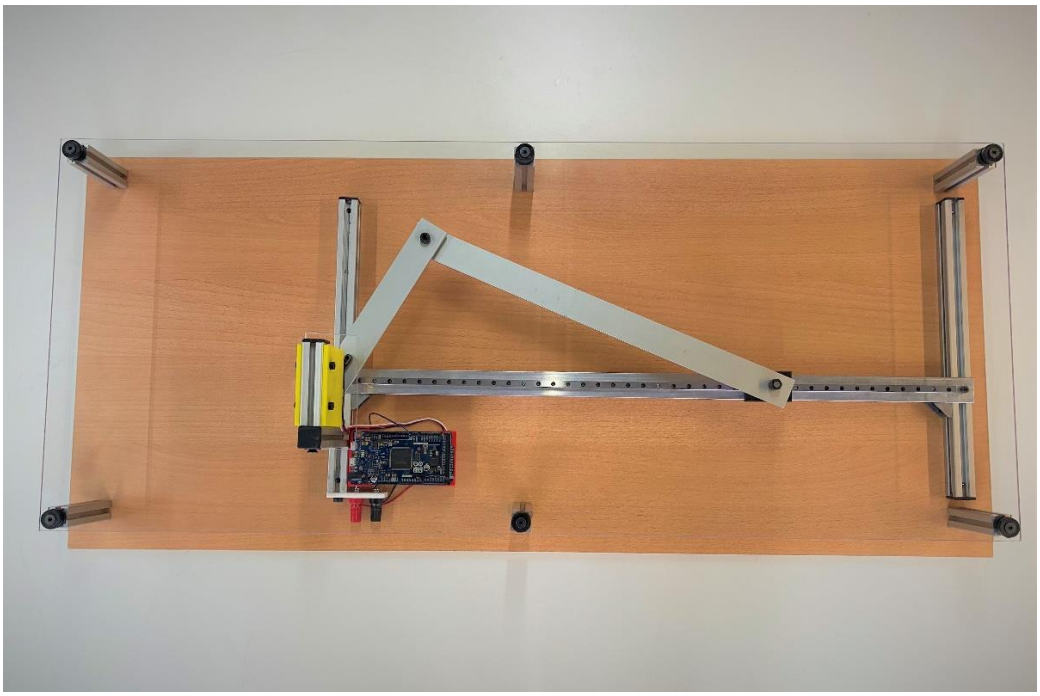


Figura 40 *Mecanisme acabat*

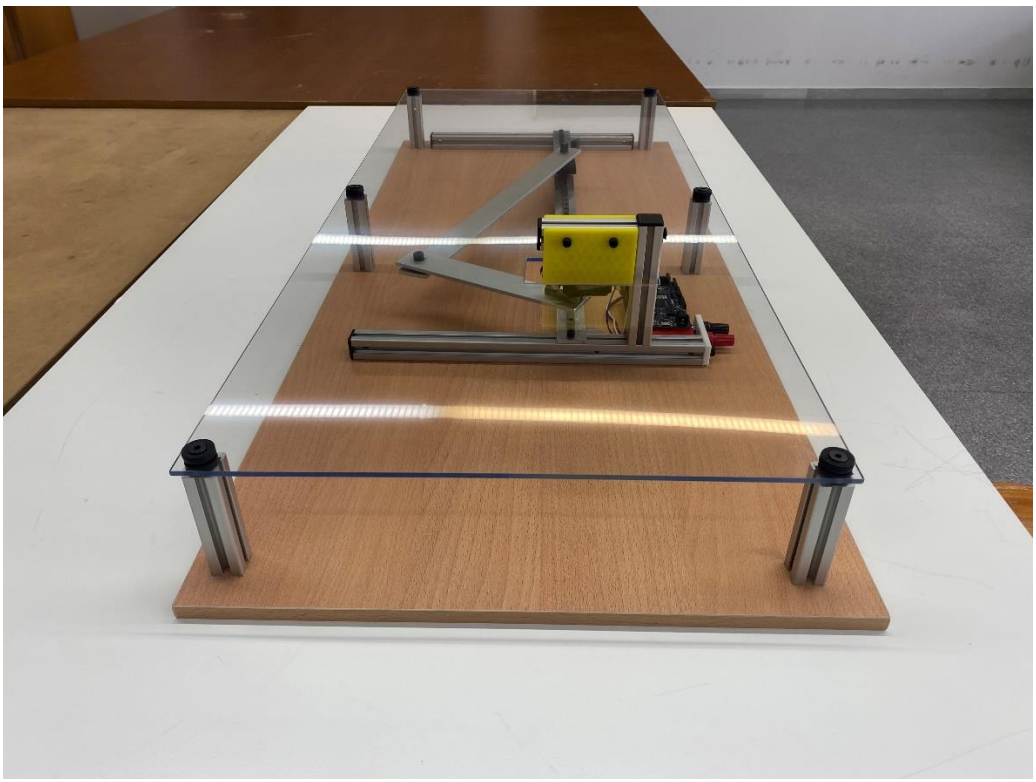


Figura 41 *Mecanisme acabat*

5. Cost del projecte

En aquest apartat s'explica el cost dels materials que s'han utilitzat i el cost total del projecte. Per a calcular el cost de les peces obtingudes per impressió 3D, com que totes les peces han estat fetes amb PLA, s'ha buscat el preu mitjà d'un quilogram d'aquest material, el qual s'ha considerat de 20€/kg. Així que considerant el preu per unitat de massa, s'ha pogut calcular el cost gràcies a les dades de massa de material necessari per imprimir cada peça que es pot aconseguir mitjançant els softwares d'impressió 3D. Cal dir que no s'ha tingut en compte el preu de l'electricitat ni l'amortització de la impressora, ja que en tractar-se de peces petites, els temps d'impressió de totes les peces en total no arriba a les tres hores, per tant, té un efecte quasi insignificant.

Com tot projecte, s'hi ha invertit hores les quals tenen un preu. Per fixar un preu/hora s'ha fet recerca sobre un preu hora mitjà d'un enginyer i s'ha trobat que correspon a uns 14€/h. Per calcular el preu associat a les hores invertides, s'ha considerat només les hores invertides en el procés de mecanitzat i construcció, ja que la resta, per repetir el projecte no caldria invertir-les, tan sols posar el codi prèviament fet a la placa controladora. Així que en el procés de mecanitzat i construcció, s'ha considerat que s'ha tardat un total de 25h. La Taula 4 resumeix el cost del projecte d'acord amb les especificacions explicades anteriorment.

Taula 4 *Cost del projecte*

Component	Quantitat	Preu (€)
Perfil alumini 20x20mm	1.9m	29
Guia Hiwin	1	6
Patí Hiwin	1	6
Placa Alumini 640x25x3mm	1	10
Biela	1	6
Manovella	1	5
Gruix biela-manovella	2	0,04
Arduino Due	1	36,8
Servomotor	1	25
Metacrilat 1000x600x4mm	1	18
Suport biela patí	1	0,12
Femelles M4 perfil 20x20mm	10	9,5
Escaires perfil alumini 20x20mm	2	8
Cargols M4	10	1
Cargols M5	17	2
Femelles M5	5	0,85
Volanderes de goma M5	12	2,5
Suport Servomotor	2	0,44
Suport Arduino Due	1	0,22
Suport connexions	1	0,08
Placa melamina 900x410x10mm	1	10
Cost hores	25	350
Total		526,55

Cal dir que en ser la primera vegada que es fa el prototip i en el fet d'haver de provar com és la millor forma de fer cada part, el temps invertit en hores és molt més gran del que es tardaria a tornar a construir el mecanisme, per la qual cosa el cost del prototip baixaria significativament. Es considera que per tornar a construir el prototip no es tardaria més de 10 hores, per la qual cosa, el cost es veuria reduït al voltant de 200€. Un altre factor a tenir en compte en la reducció de costos és que si fos el cas de fabricació en massa d'aquest mecanisme el preu del material, en poder comprar-lo a l'engròs, es reduiria significativament.

6. Manual d'instruccions d'ús

En aquest apartat s'explica detalladament els passos que s'han de seguir per utilitzar correctament el mecanisme. El material necessari per fer una simulació es veu resumit a la Taula 5.

Taula 5 *Material necessari per simular*

Material	Quantitat	Especificacions
Font d'alimentació	1	S'ha de configurar amb un voltatge de 6.8V i un corrent limitat a 3A o 4A aproximadament.
Cables font d'alimentació	2	Cables per fer l'alimentació del mecanisme des de la font d'alimentació fins als pins de connexió del mecanisme.
Ordinador	1	Ordinador amb l'aplicació instal·lada per entrar les dades.
Cable USB-MicroUSB	1	Cable per poder connectar l'Arduino a l'ordinador.

Una vegada es disposa de tot el material mostrat en la taula anterior, seguidament, s'explica detalladament per punts els passos que s'han de seguir per fer funcionar el mecanisme de forma correcta.

1. Connectar el cable USB procedent de l'Arduino Due a un port USB de l'ordinador que s'usarà per entrar les dades de la simulació.

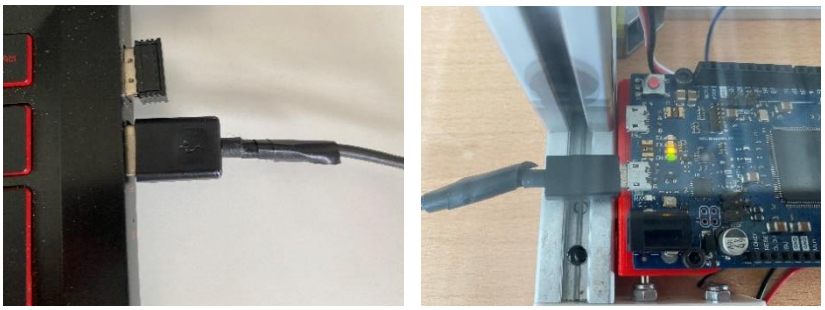


Figura 42 *Connectar Arduino amb PC*

2. Una vegada connectada la placa a l'ordinador, es procedeix inicialitzant l'aplicació prèviament instal·lada.

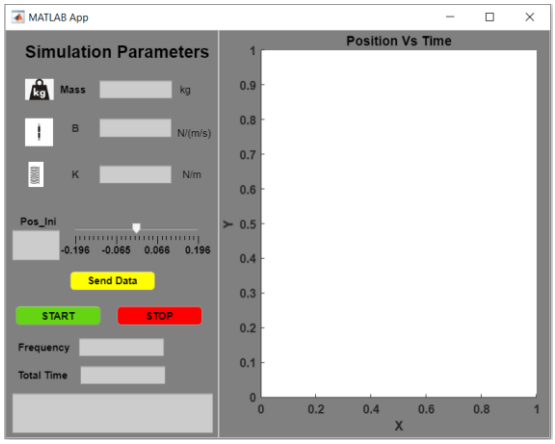


Figura 43 *Aplicació inicialitzada*

3. Configurar la font d'alimentació amb una tensió de 6.8V i un corrent màxim d'entre 3A i 4A.

Figura 44 *Configuració font d'alimentació*

4. Connectar els cables de la font d'alimentació als port indicats al mecanisme, sempre tenint en compte que el connector negre correspon al negatiu i el vermell al positiu.

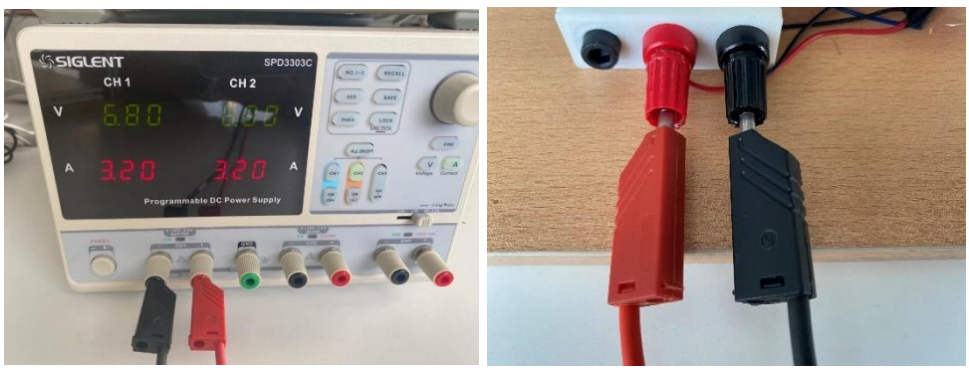


Figura 45 *Connexió font d'alimentació i mecanisme*

5. Una vegada està alimentat i s'han complert els passos anteriors ja es pot procedir a entrar les dades que es desitgen per a fer la simulació. Per fer ús de l'aplicació cal fer-ho amb el següent ordre.
    1. Entrar el valor de la massa en kg.
    2. Entrar el valor de la constant d'esmorteïment B.
    3. Entrar el valor de la constant de la molla K.
    4. Moure el *slider* per decidir la posició inicial del mecanisme.
    5. Prémer el botó *Send Data*.
    6. Llegir el missatge de la simulació.
    7. Prémer *Start* si vols seguir amb la simulació o prémer *Stop* i tornar a seguir els passos dels del primer punt.
    8. Una vegada s'estigui simulant, si es vol parar la simulació cal prémer *Stop* i si es vol fer una altra simulació es tornen a seguir els passos o es pot tancar l'aplicació.

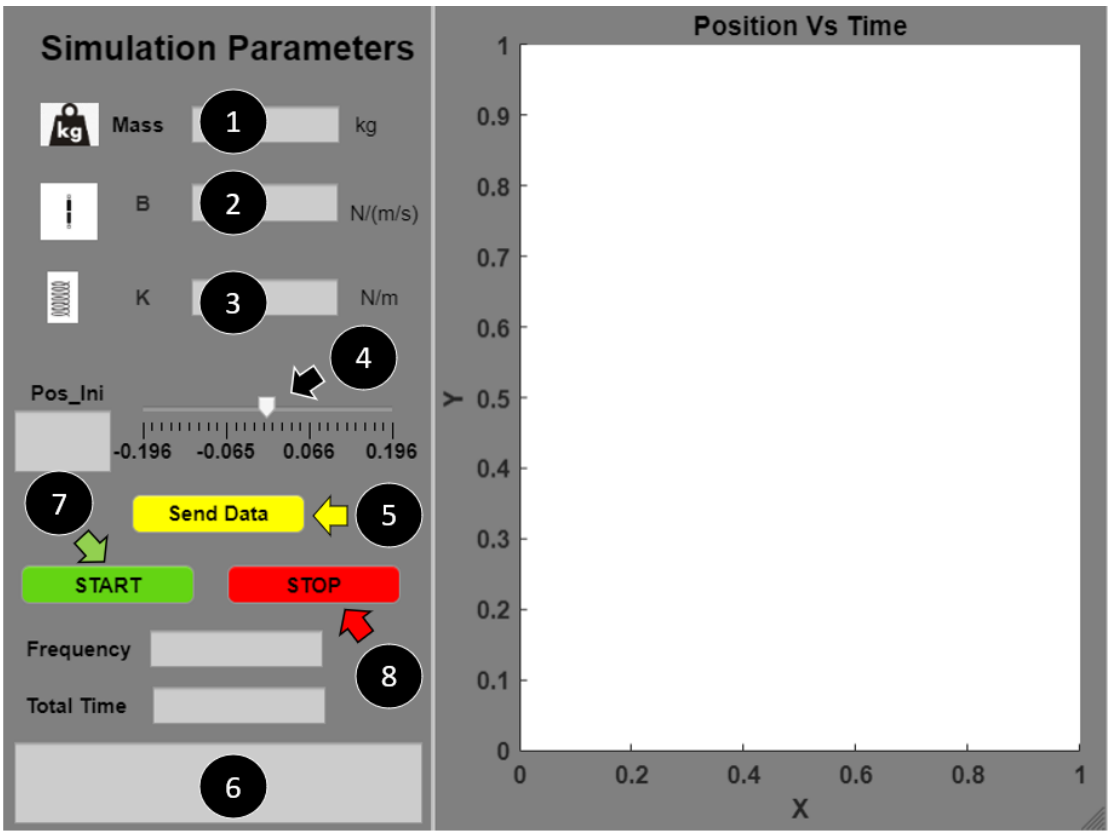


Figura 46 *Passos a seguir per entrar dades*

7. Discussió de resultats i conclusions

Durant el grau d'enginyeria mecatrònica s'han dut a terme assignatures de disseny, de control, de programació, de càlcul i és que en aquest treball s'ha tocat una petita part de cadascuna de les assignatures que s'han treballat durant els darrers quatre anys, per la qual cosa ha servit per fer memòria d'algunes parts que havien quedat més borroses i, al mateix temps, aprofundir més alguns conceptes.

Durant el procés del Treball de Final de Grau, ha sigut crucial l'organització del temps i complir el calendari fet a l'inici del projecte, el qual hi havia indicat quants dies calia destinar com a màxim a cada part del projecte per tal de poder completar tots els objectius que es tenien per aquest treball.

Per a discutir els resultats obtinguts amb el projecte cal avaluar el grau del compliment dels objectius proposats a l'inici del projecte. El primer objectiu fa referència a fer un mecanisme segur que sigui un recurs pedagògic. Aquest propòsit es pot donar per assolit, ja que s'ha construït el mecanisme i, a part, se li ha instal·lat unes proteccions per fer-lo segur. La part de fer que sigui un recurs pedagògic és un aspecte que caldria posar-ho en pràctica per tenir evidències de si resulta útil pels alumnes. Així que un aspecte a millorar en aquesta part seria la posada en pràctica del prototip amb alumnes per tenir evidències de si és un recurs educatiu útil que els permet comprendre i entendre de forma més visual com evolucionen les oscil·lacions d'aquests tipus.

En referència a la part econòmica, el cost del mecanisme ha estat per sota dels 600 €, és per això que es pot donar per complert l'objectiu de fer un mecanisme econòmic, entenent per econòmic un cost inferior a 1000 €, gràcies a la tria de components i materials adequats.

A continuació s'avalua de forma més detallada l'àmbit mecànic i de programació.

Àmbit mecànic

Com a resultats mecànics, cal avaluar el compliment dels objectius que fan referència al disseny, a la mecanització i a la construcció del prototip. L'objectiu referent a fer ús del mecanisme biela-manovella es pot donar per complert, ja que tal com s'explica en la memòria, hi ha altres alternatives, però per complir aquest requisit, s'utilitza aquest sistema adaptant el disseny mecànic i l'actuador per transformar el moviment circular que aquest proporciona en un moviment lineal.

Un objectiu del mecanisme fa referència a aconseguir que el prototip sigui lleuger, robust i manejable. Es podrien donar per assolits tots els objectius que es proposen en aquest apartat sobretot en l'apartat de lleuger i robust, ja que el mecanisme és robust i el seu pes és de 5.5 kg aproximadament contant-hi tota l'electrònica, la protecció i tots els components. Un aspecte no tant satisfactori referent a aquest objectiu és la manejabilitat, ja que és un mecanisme gran i pot suposar complicacions a l'hora de transportar-lo.

El disseny del mecanisme s'ha fet totalment, de cada una de les peces utilitzades se n'ha fet el CAD i el plànol corresponent. A més, s'ha fet l'assemblatge amb tots els components que formen el mecanisme, és per això, que l'objectiu de dissenyar es pot donar per assolit tal com es pot apreciar a la Figura 20, Figura 21 i Figura 22.

La mecanització i construcció es poden donar per aconseguits els objectius d'aquestes parts, ja que tal com es pot veure en la Figura 40 i Figura 41, s'han mecanitzat totes les peces que ha calgut i s'han muntat per garantir el correcte funcionament.

En referència a l'objectiu de complir requeriments d'amplitud i freqüència, cal dir que ambdós es poden donar per complerts, ja que el mecanisme permet fer oscil·lacions amb una amplitud de quasi 0.4 metres i freqüències de fins a 1.2 Hz aproximadament. Si es desitgés tenir un rang de possibilitats de simulació més gran, un punt de millora que seria interessant és fer ús d'un servomotor més potent i més ràpid, ja que d'aquesta manera, en augmentar la velocitat permetria simular oscil·lacions amb una freqüència més alta. Per altra banda, el fet d'utilitzar un servomotor més potent pel

que fa a especificacions de parell, permetria que el mecanisme es pogués penjar en vertical per veure les oscil·lacions.

Àmbit de programació

En l'apartat de programació, en aquest projecte s'ha programat el mecanisme amb el Simulink i amb l'Arduino IDE, per la qual cosa els objectius relacionats amb la programació mitjançant aquests programaris i fent que el sistema sigui accessible per a tothom, es poden donar per assolits, ja que el software que fa falta és gratuït i el pot tenir quasi tothom.

L'objectiu específic relacionat en fer una aplicació simple, visual i entenedora, cal dir que també es pot donar per assolit, ja que tal com s'ha vist, s'ha pogut desenvolupar una aplicació amb l'App Designer molt intuïtiva, simple i visual, tal com es mostra a la Figura 31.

Fer aquest projecte m'ha permès aprendre sobretot tasques de programació, comunicació entre dispositius, disseny de peces, paràmetres importants d'impressió 3D, conèixer softwares que es desconeixien o no es creia que tingués el potencial que realment tenen, entre d'altres. És el cas del Matlab, aquest programari m'ha sorprès gratament, ja que tal com s'ha vist en aquest projecte, s'ha fet servir per fer càlculs, programació, comunicació entre l'ordinador i l'Arduino i, inclús, per fer una aplicació d'ordinador.

Des del meu punt de vista, destaco la part de disseny i construcció del mecanisme perquè personalment, penso que ha quedat força agradable a la vista, és funcional i compleix els objectius que aquest ha de complir.

En conclusió, aquest treball, l'objectiu principal del qual era fer un dispositiu per simular el moviment oscil·latori dels sistemes formats per masses, molles i esmorteïdors, ha servit per desenvolupar una maqueta que possiblement pot ajudar a futurs alumnes a entendre millor com evolucionen les oscil·lacions al llarg del temps i, com a part personal, a aprendre molts coneixements que s'han anat adquirint durant la realització d'aquest projecte.

Bibliografia

Alibaba. (s.d). Guía lineal de alta precisión. Figura 5. https://spanish.alibaba.com/product-detail/High-Precision-Linear-Guide-Slide-X-62384520036.html

Annimos. (s.d). ANNIMOS Servo Digital. Figura 17. https://cutt.ly/GJkZTTK

Arduino. (2022). Serial communication. https://www.arduino.cc/reference/en/language/functions/communication/serial/

Arduino. (2022). Serial write. https://www.arduino.cc/reference/en/language/functions/communication/serial/write/

Arduino. (2022). Serial read. https://www.arduino.cc/reference/en/language/functions/communication/serial/read/

Arduino. (2015). Arduino Due Front. Figura 16. https://commons.wikimedia.org/wiki/File:ArduinoDue_Front.jpg

Dademuchconnection. (2017). Dinámica de un Sistema Masa-Resorte-Amortiguador. https://dademuch.com/2017/07/18/dinamica-de-un-sistema-masa-resorte-amortiguador/

Enciclopedia. (s.d). Amortidor. https://www.enciclopedia.cat/gran-enciclopedia-catalana/amortidor

Free aptitude camp. (s.d). Velocity Analysis of Silder Crank Mechanism (graphical method).https://www.freeaptitudecamp.com/velocity-analysis-slider-crank-mechanism-graphical-method/

Gabriel. (s.d) Assesoría técnica. Funcionamento del amortiguador. https://www.gabriel.com.mx/asesoria-tecnica/funcionamiento-del-amortiguador/

Getauto. (2017). Sistema de suspensión del coche. Figura 4. https://getauto.es/sistema-de-suspension/

Geauto. (s.d). Sistema de suspension del coche. https://getauto.es/sistema-de-suspension/

MATLAB. (2022). MATLAB App Desinger. https://www.mathworks.com/products/matlab/app-designer.html

Ogata, Katsuhiko. (2010). *Ingeniería de control moderna.* (5a. ed.). Pearson.

Russell, D. (2018). An Example of a Non-Harmonic Oscillator: Crankshaft Motion. https://www.acs.psu.edu/drussell/Demos/crankshaft/crankshaft.html

Russell, D. (2018). An Example of a Non-Harmonic Oscillator: Crankshaft Motion. Figura 3. https://www.acs.psu.edu/drussell/Demos/crankshaft/crankshaft.html

Vikipedia. (2022). Teorema del cosinus. https://ca.wikipedia.org/wiki/Teorema_del_cosinus#/media/Fitxer:Triangle_with_notations_2.svg

Viquipedia. (2021). Esmorteïment. https://ca.wikipedia.org/wiki/Esmorte%C3%AFment

Viquipedia. (2022). Molla. https://ca.wikipedia.org/wiki/Molla

Weisman, D. (2006). Triangle with notations 2. Figura 10. https://ca.wikipedia.org/wiki/Teorema_del_cosinus#/media/Fitxer:Triangle_with_notations_2.svg

Apèndix A Plànols peces CAD

En aquest apartat de l'apèndix es troben els plànols de les peces dissenyades.

A.1 Plànol Manovella

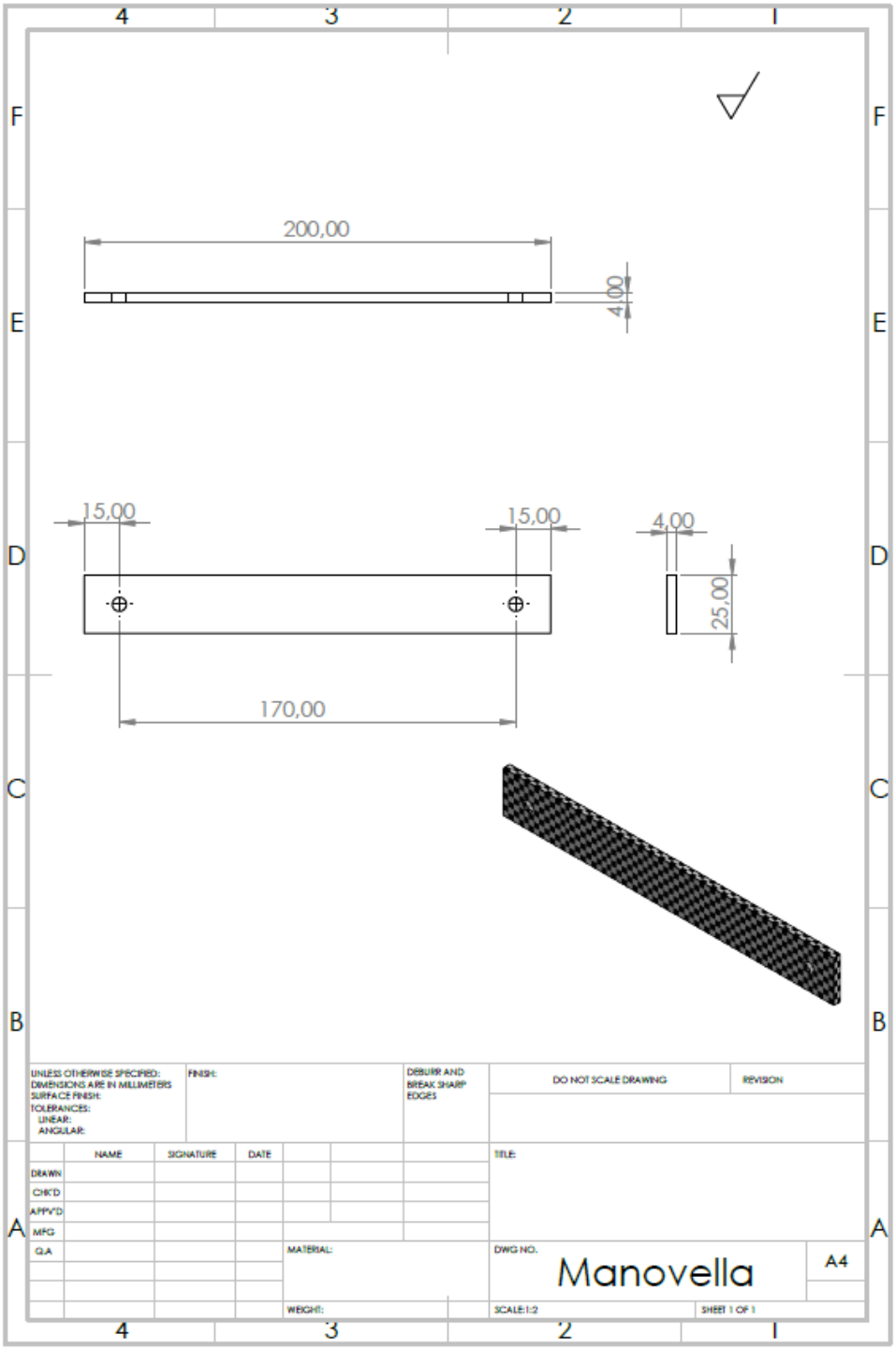


Figura Apèndix A. 1 *Plànol manovella*

A.2 Plànol biela

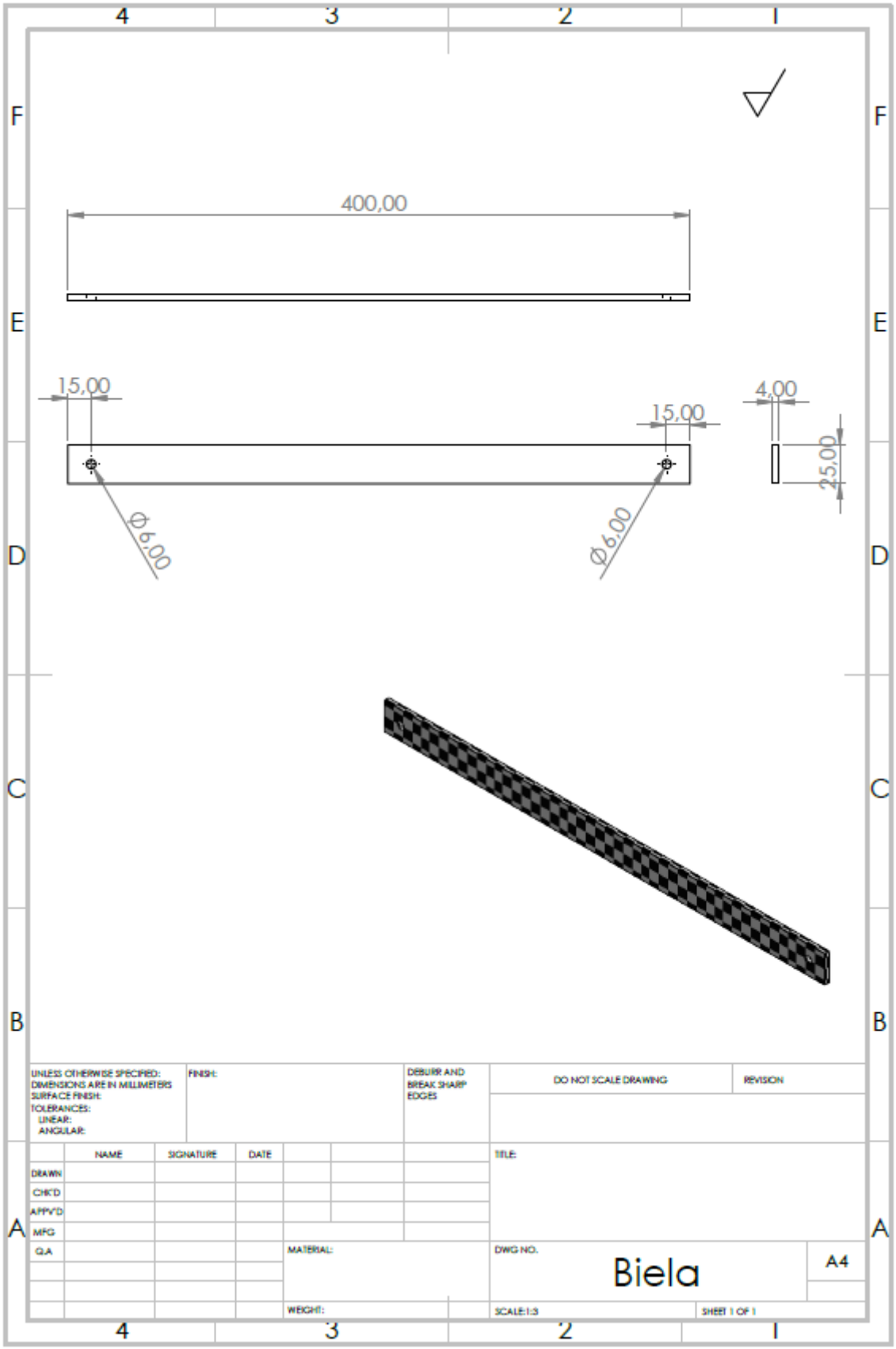


Figura Apèndix A. 2 *Plànol Biela*

A.3 Plànol guia Hiwin

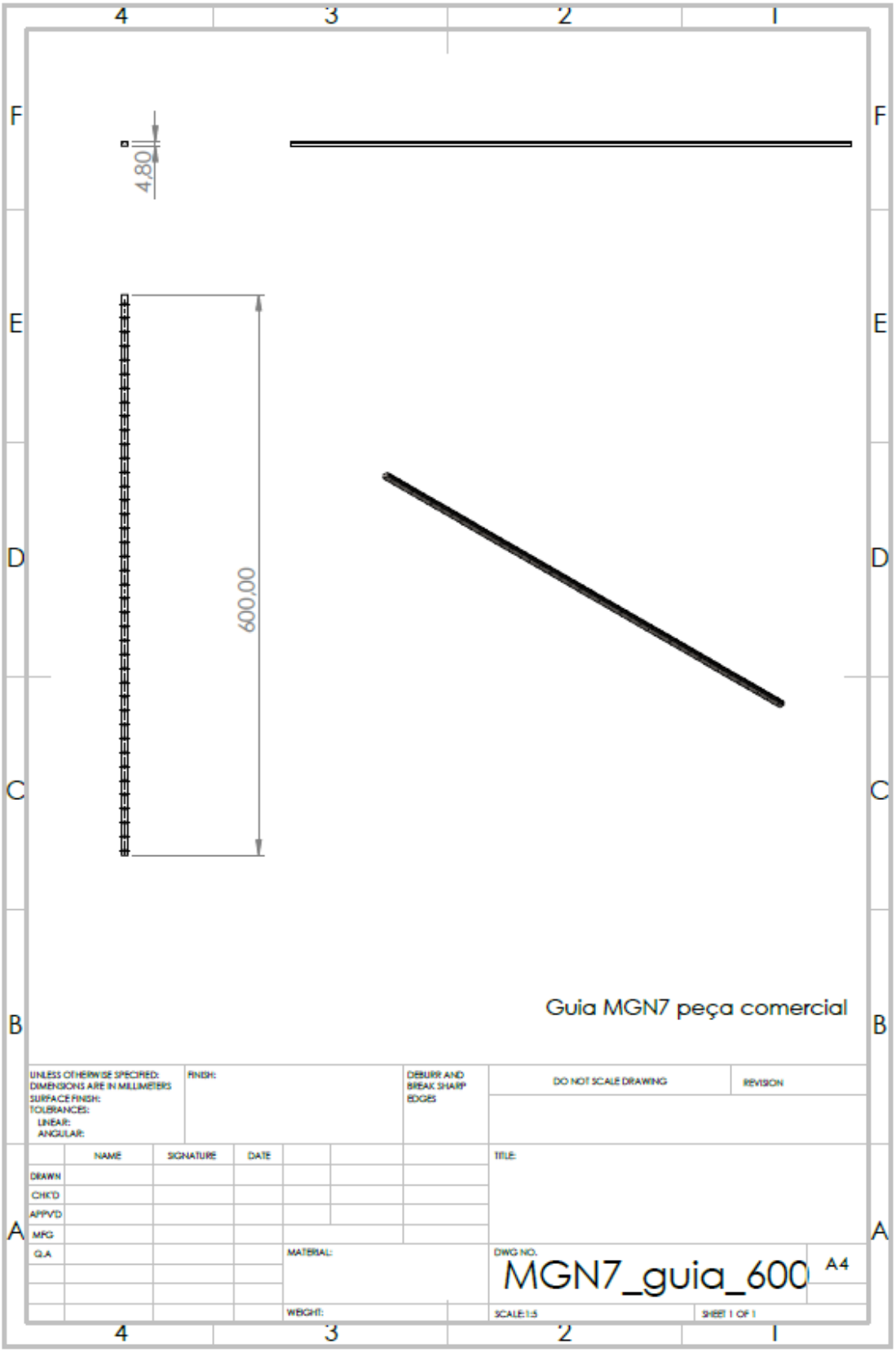


Figura Apèndix A. 3 *Plànol guia Hiwin*

A.4 Plànol suport guia Hiwin

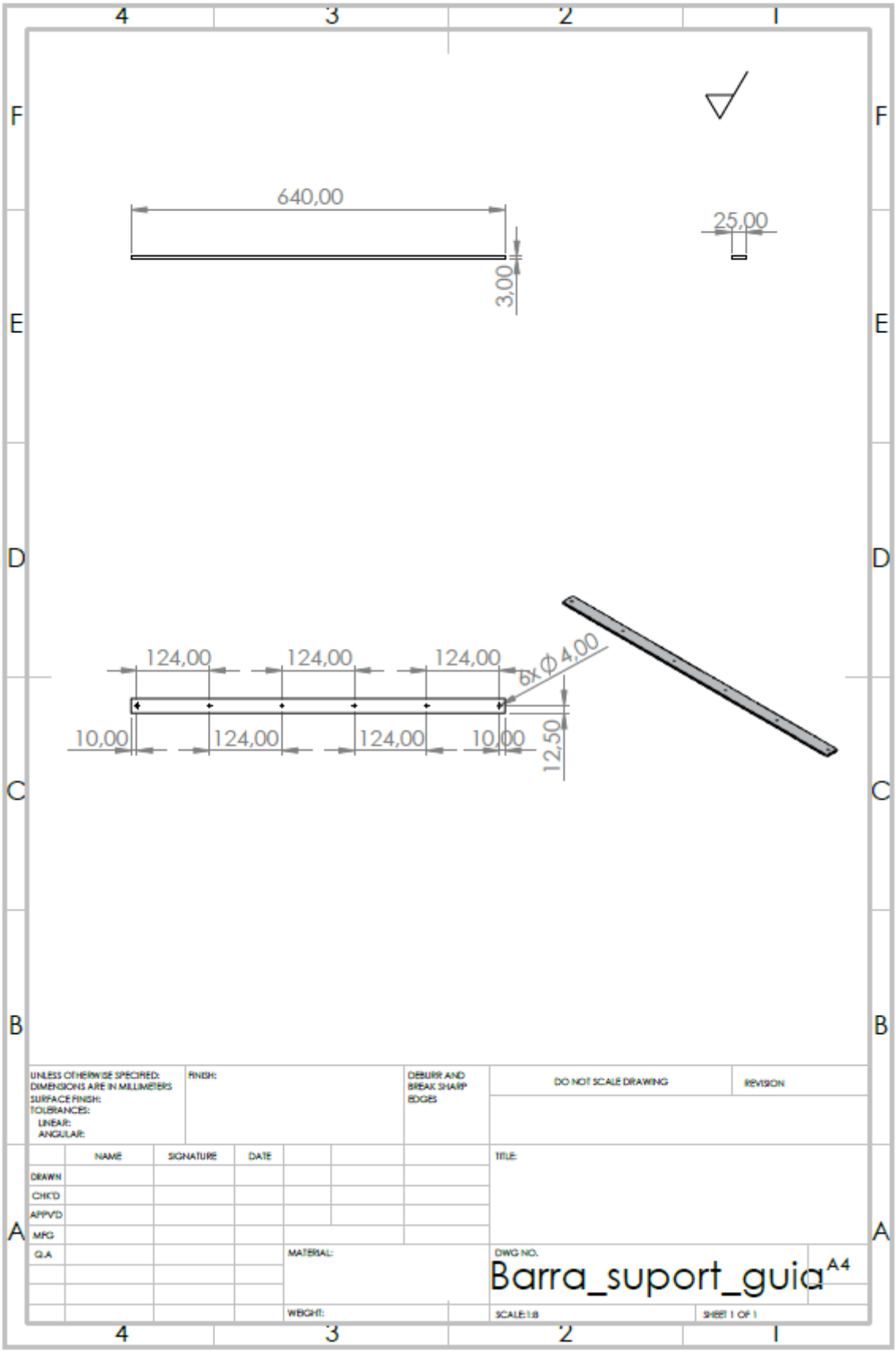


Figura Apèndix A. 4 *Plànol suport guia Hiwin*

A.5 Plànol patí Hiwin

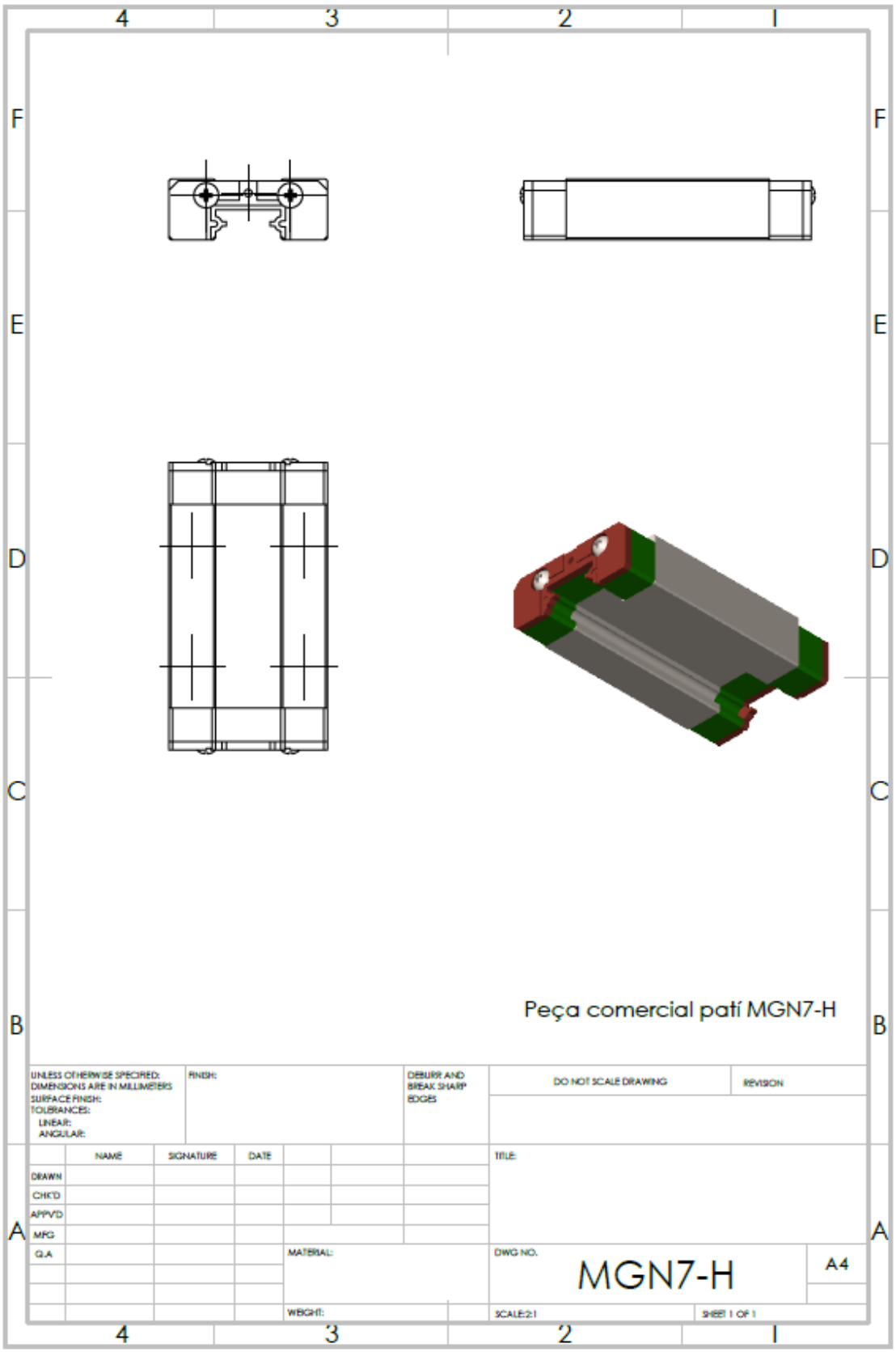


Figura Apèndix A. 5 *Plànol patí Hiwin*

A.6 Plànol suport patí biela

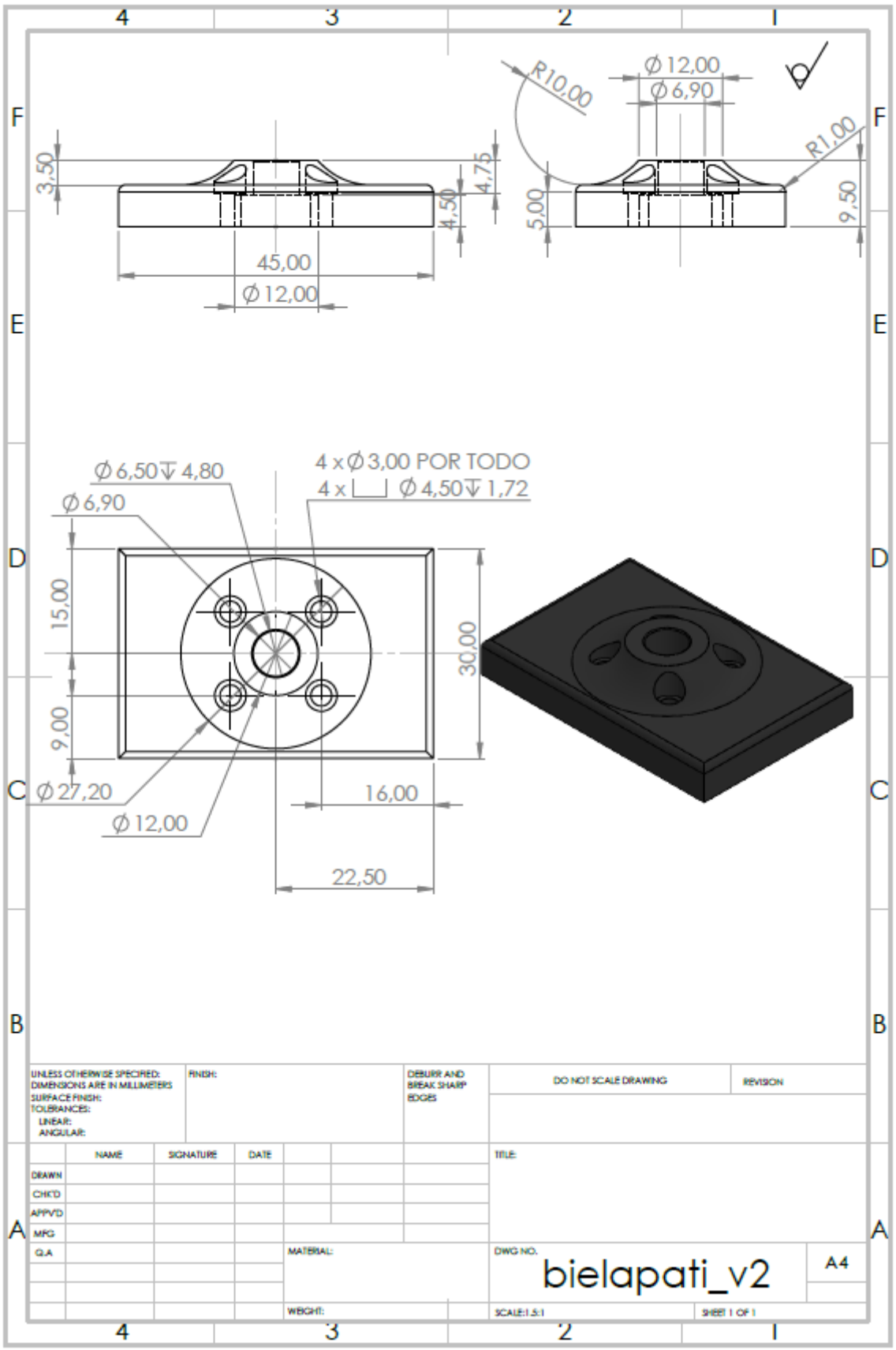


Figura Apèndix A. 6 *Plànol suport patí biela*

A.7 Plànol lateral esquerre

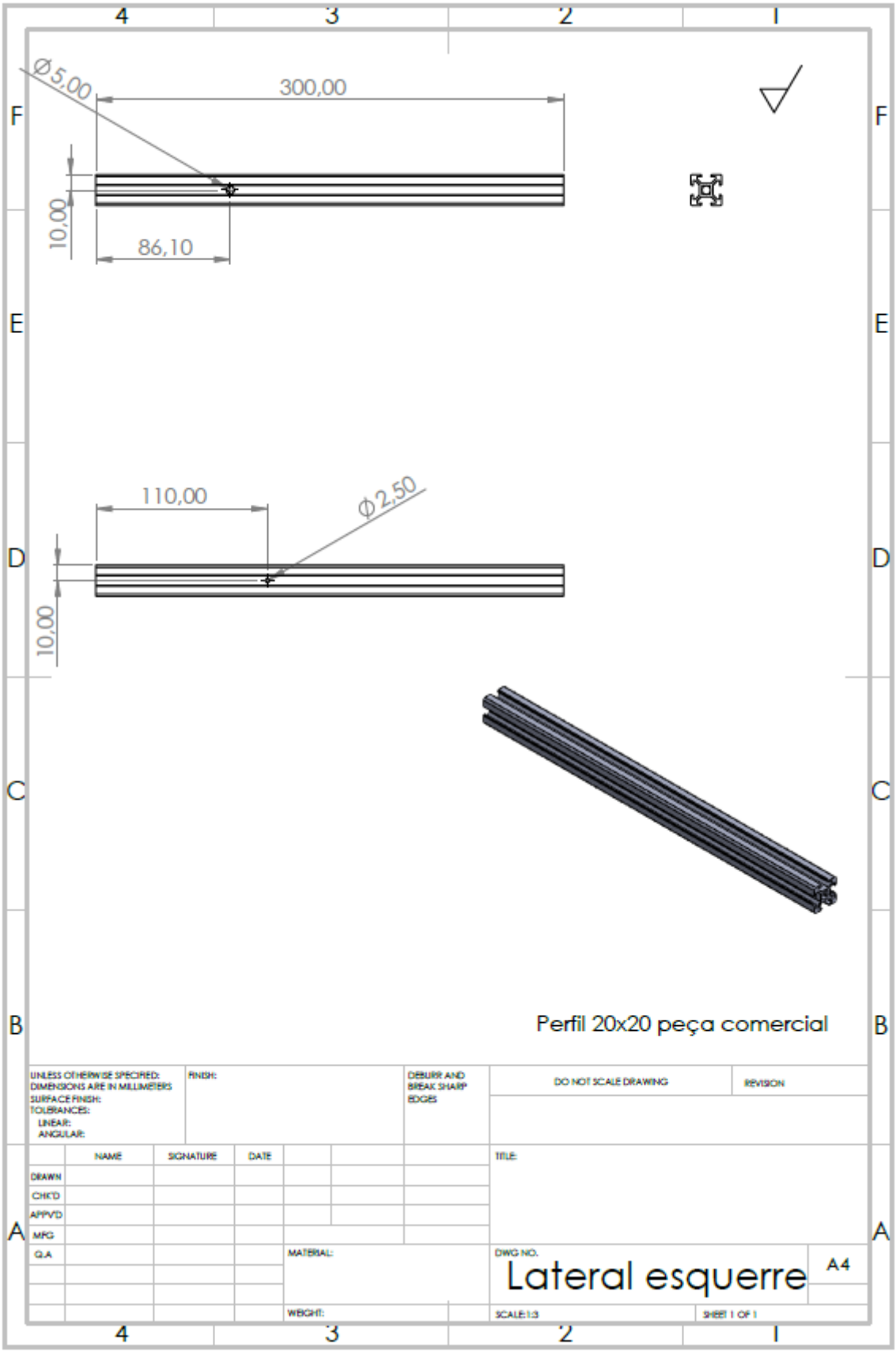


Figura Apèndix A. 7 *Plànol lateral esquerre*

A.8 Plànol lateral dret

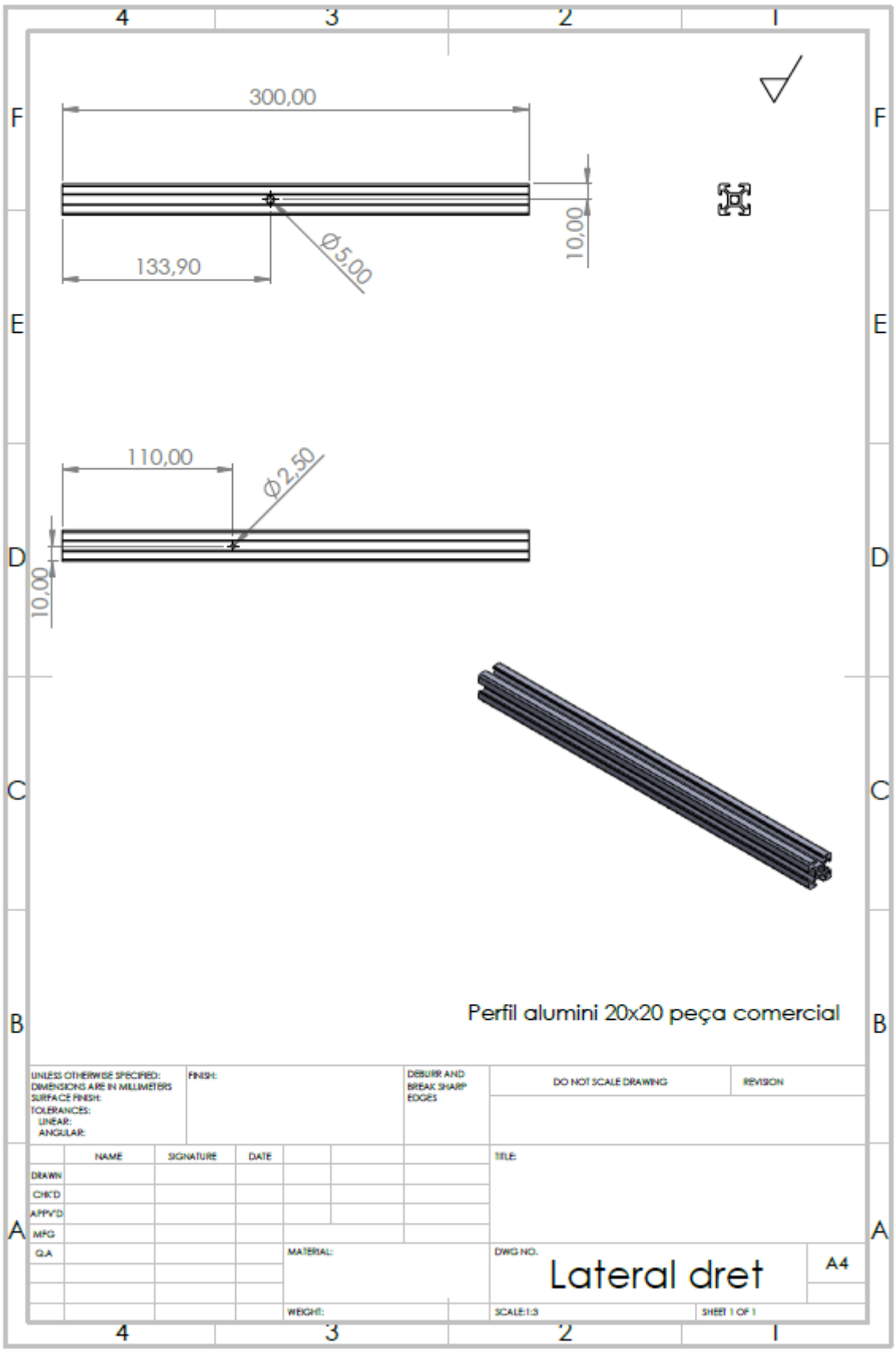


Figura Apèndix A. 8 *Plànol lateral dret*

A.9 Plànol suport horitzontal

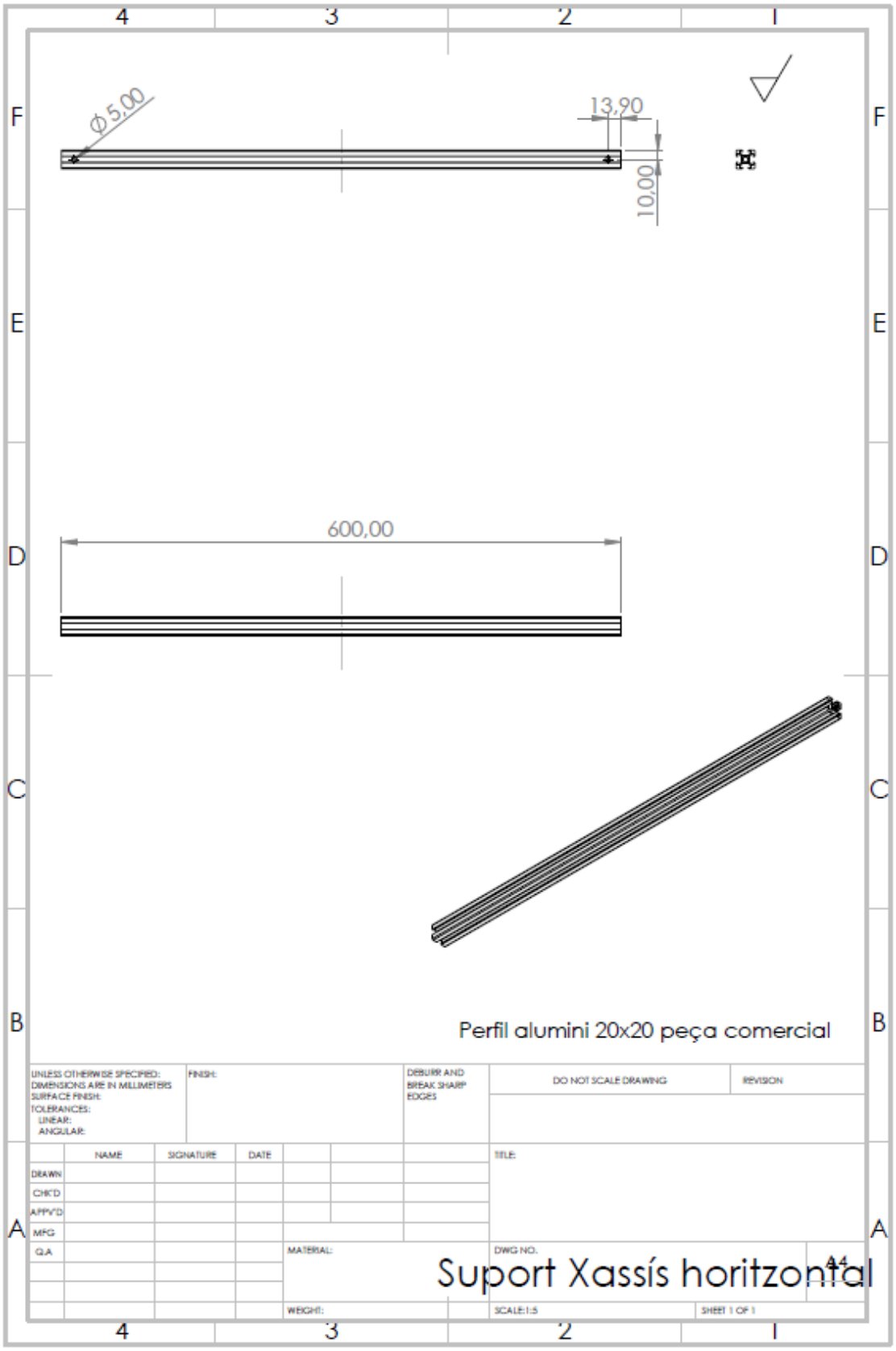


Figura Apèndix A. 9 *Plànol suport horitzontal*

A.10 Plànol suport servomotor vertical

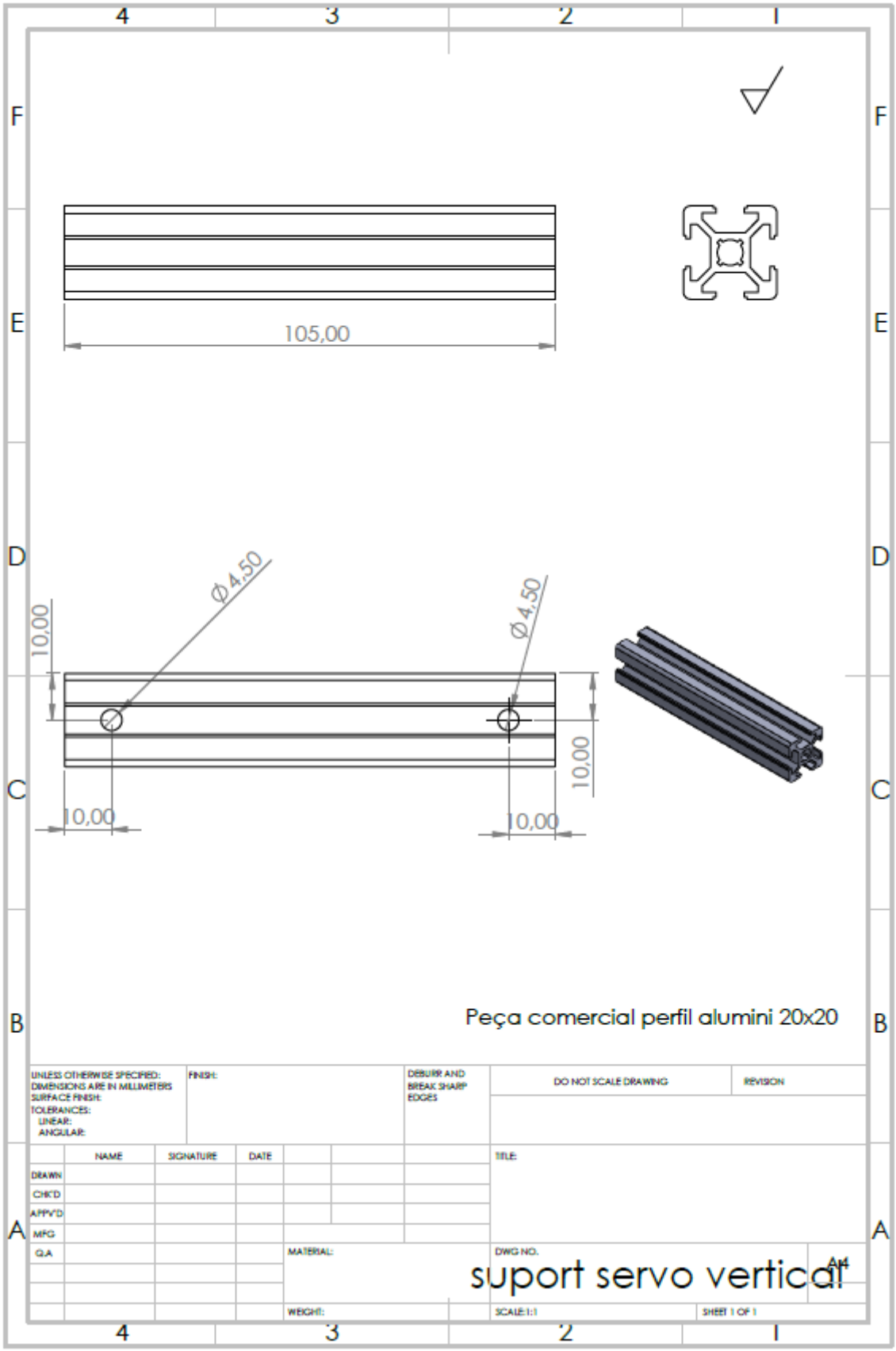


Figura Apèndix A. 10 *Plànol suport servomotor vertical*

A.11 Plànol suport servomotor horitzontal

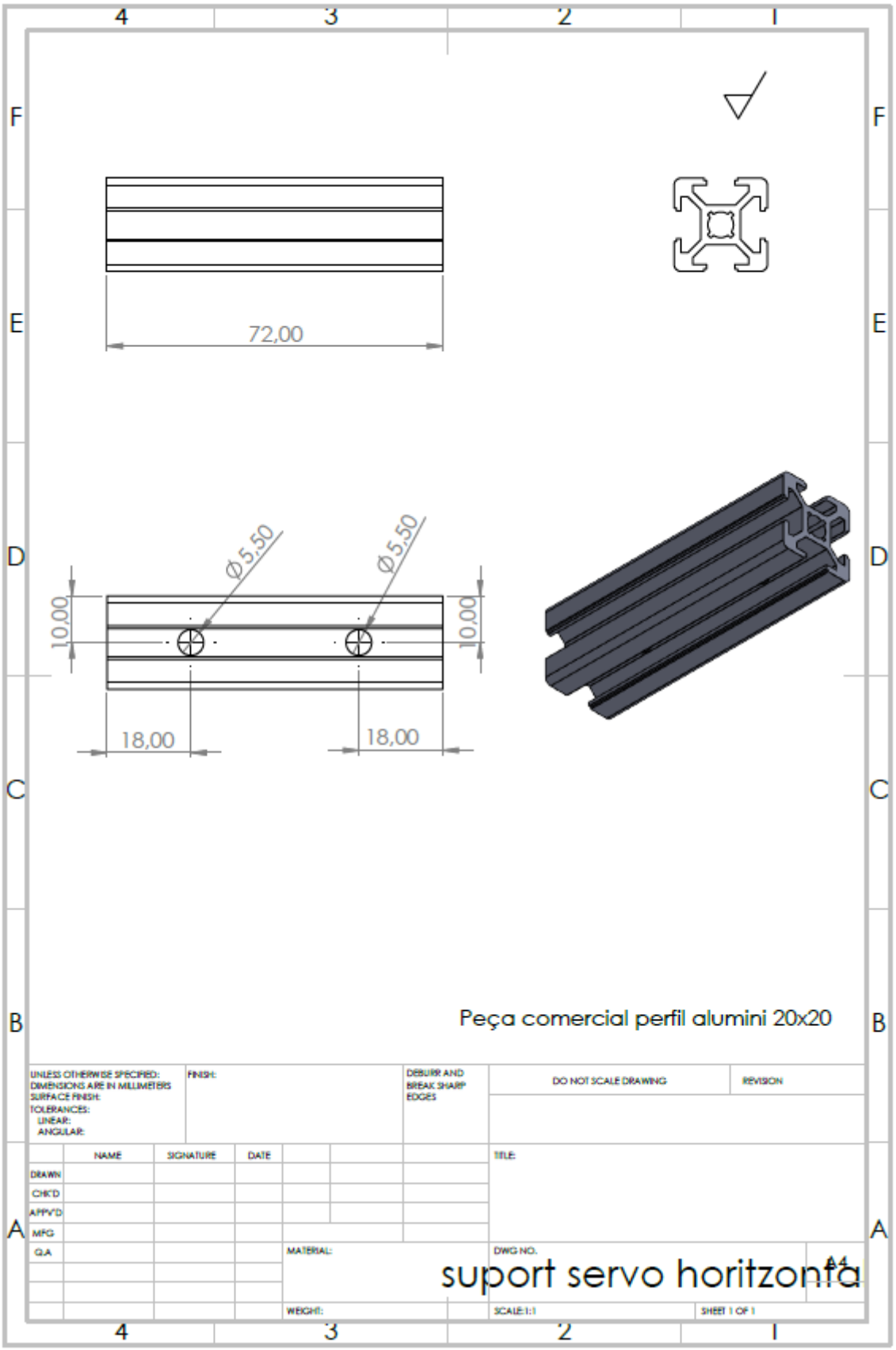


Figura Apèndix A. 11 *Plànol suport servomotor horitzontal*

A.12 Plànol suport servomotor

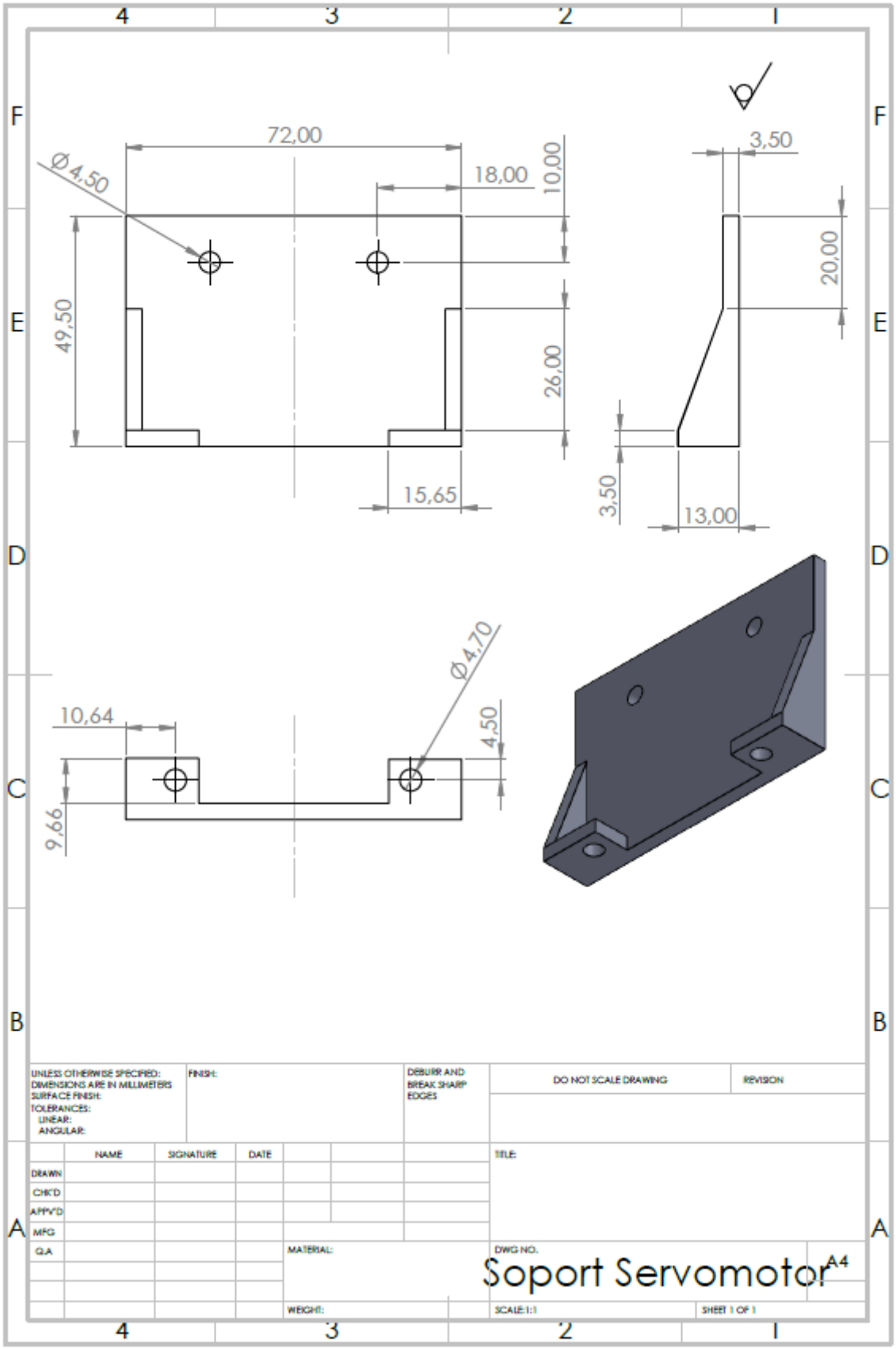


Figura Apèndix A. 12 *Plànol suport servomotor*

A.13 Plànol suport Arduino

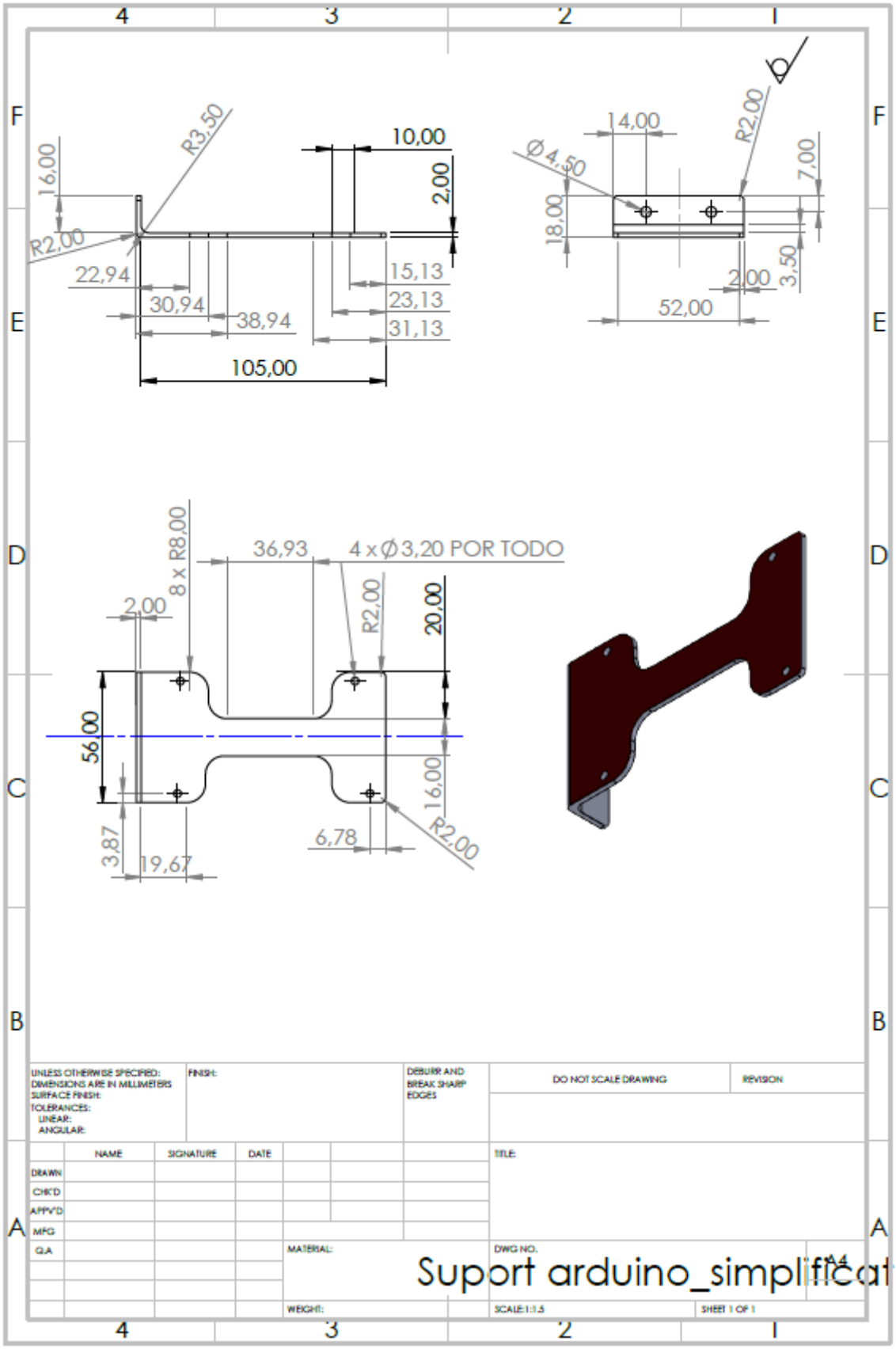


Figura Apèndix A. 13 *Plànol suport Arduino*

A.14 Plànol suport connexions

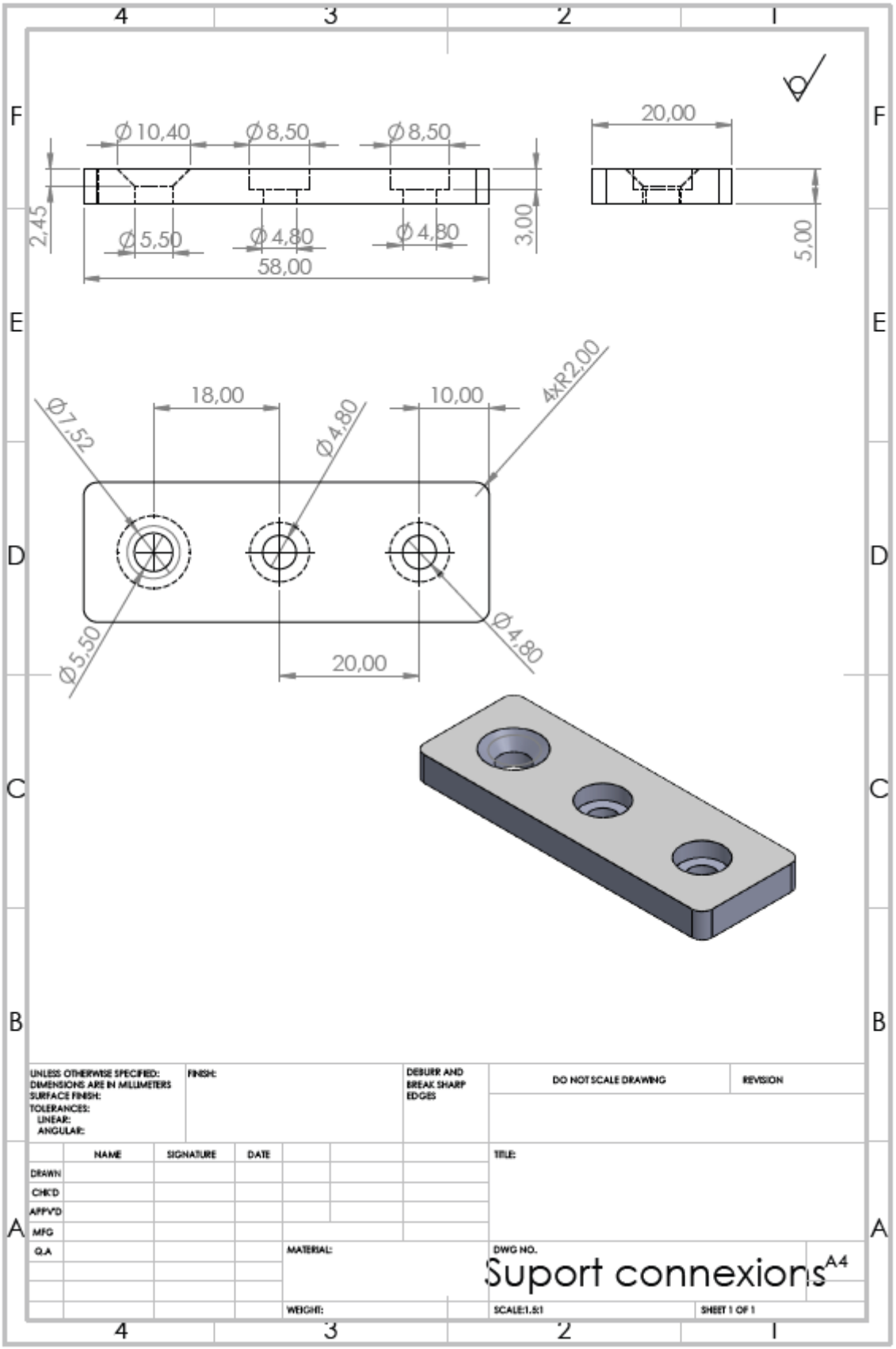


Figura Apèndix A. 14 *Plànol suport connexions*

A.15 Plànol peus protecció

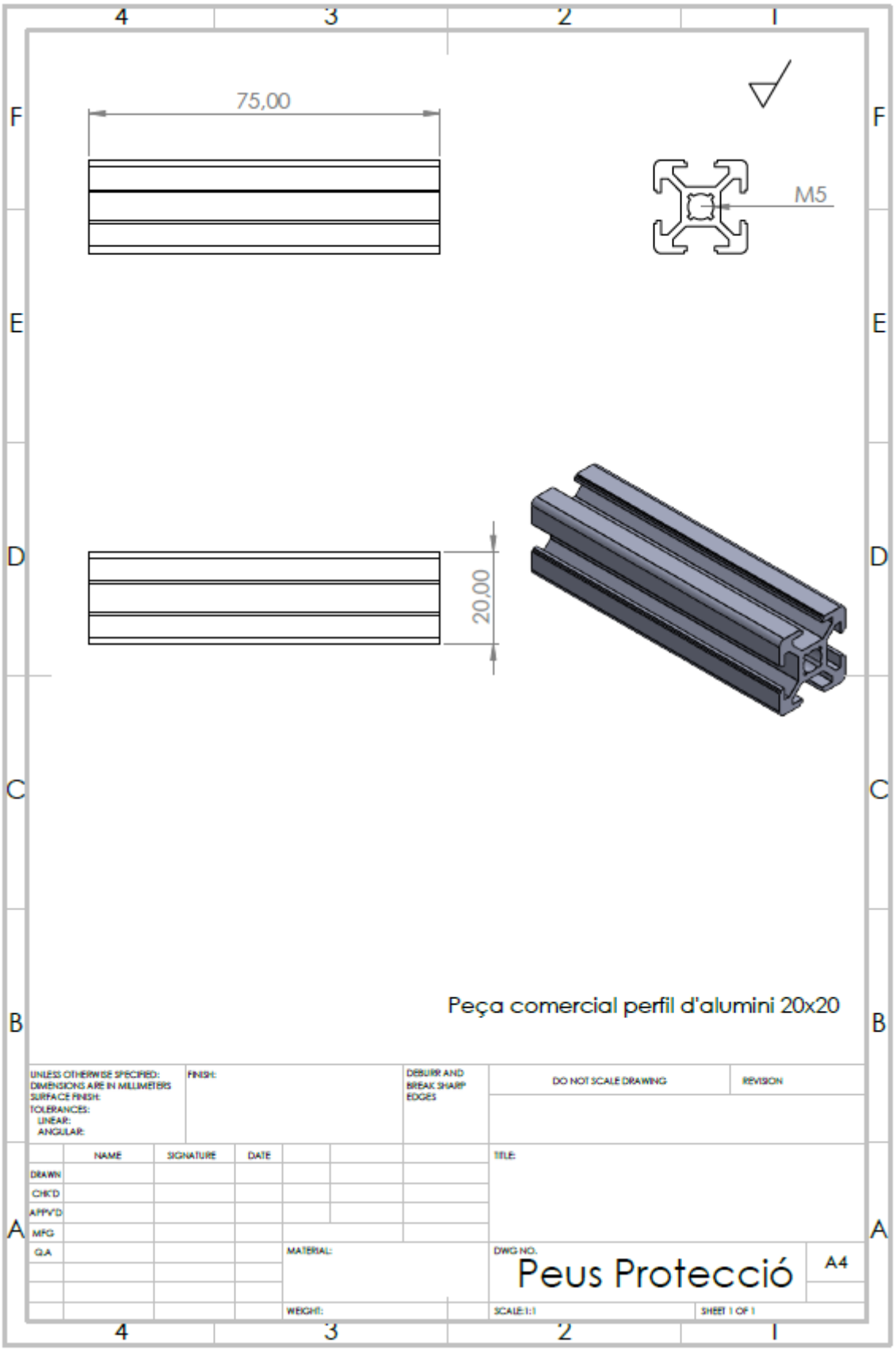


Figura Apèndix A. 15 *Plànol peus protecció*

A.16 Plànol protecció

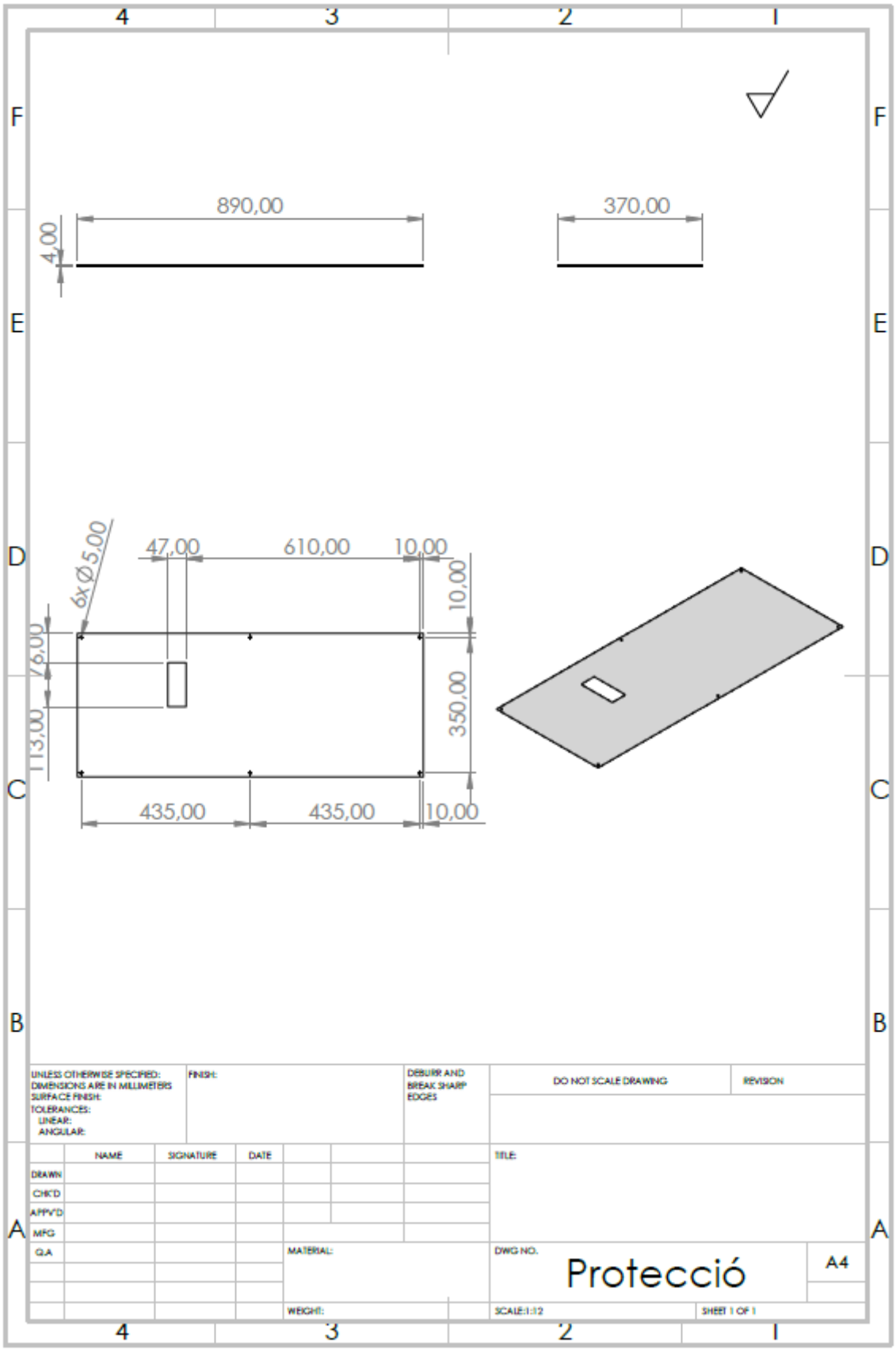


Figura Apèndix A. 16 *Plànol protecció*

A.17 Plànol gruix biela-manovella

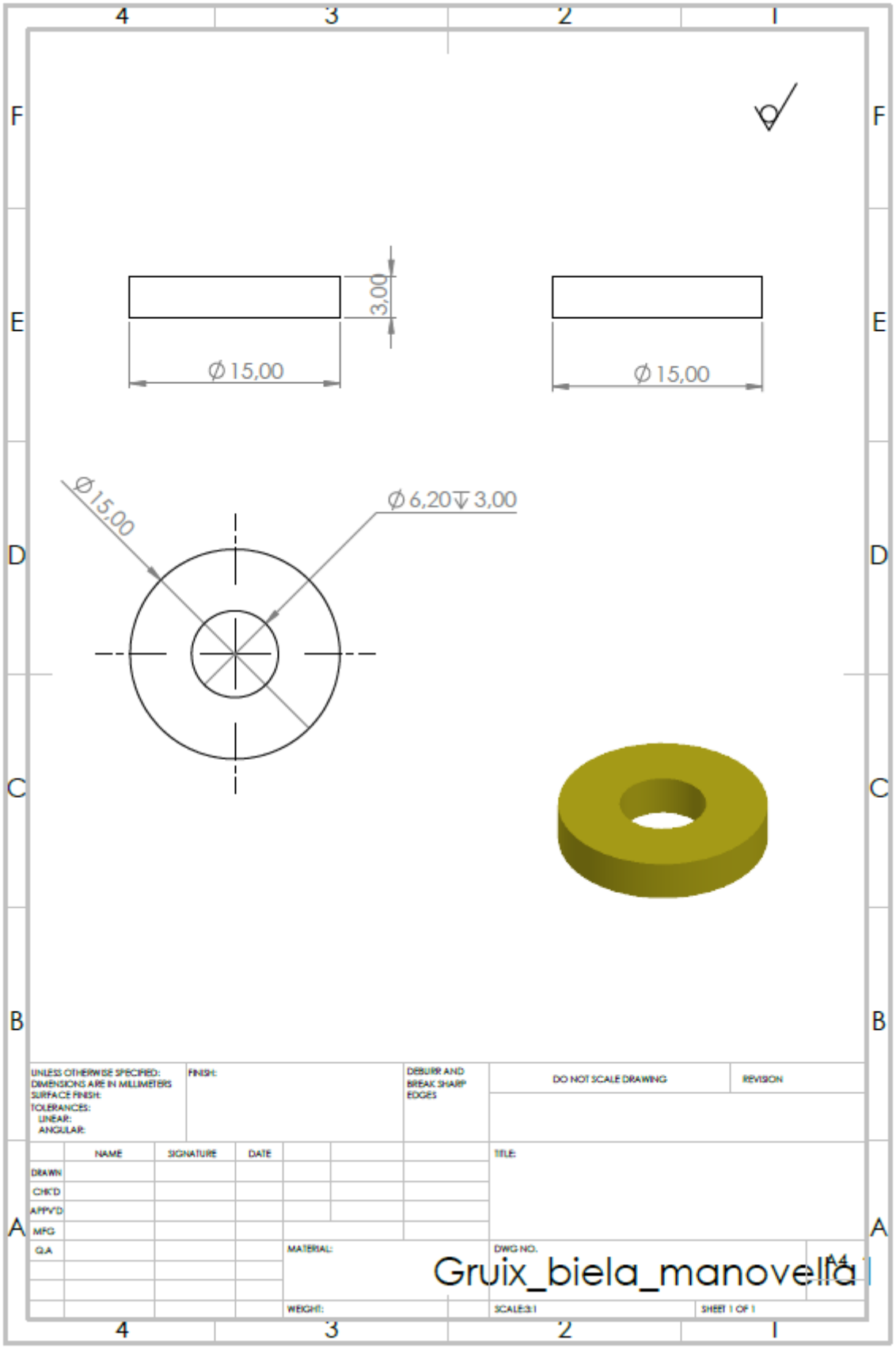


Figura Apèndix A. 17 *Plànol gruix biela-manovella*

A.18 Plànol femella perfil alumini

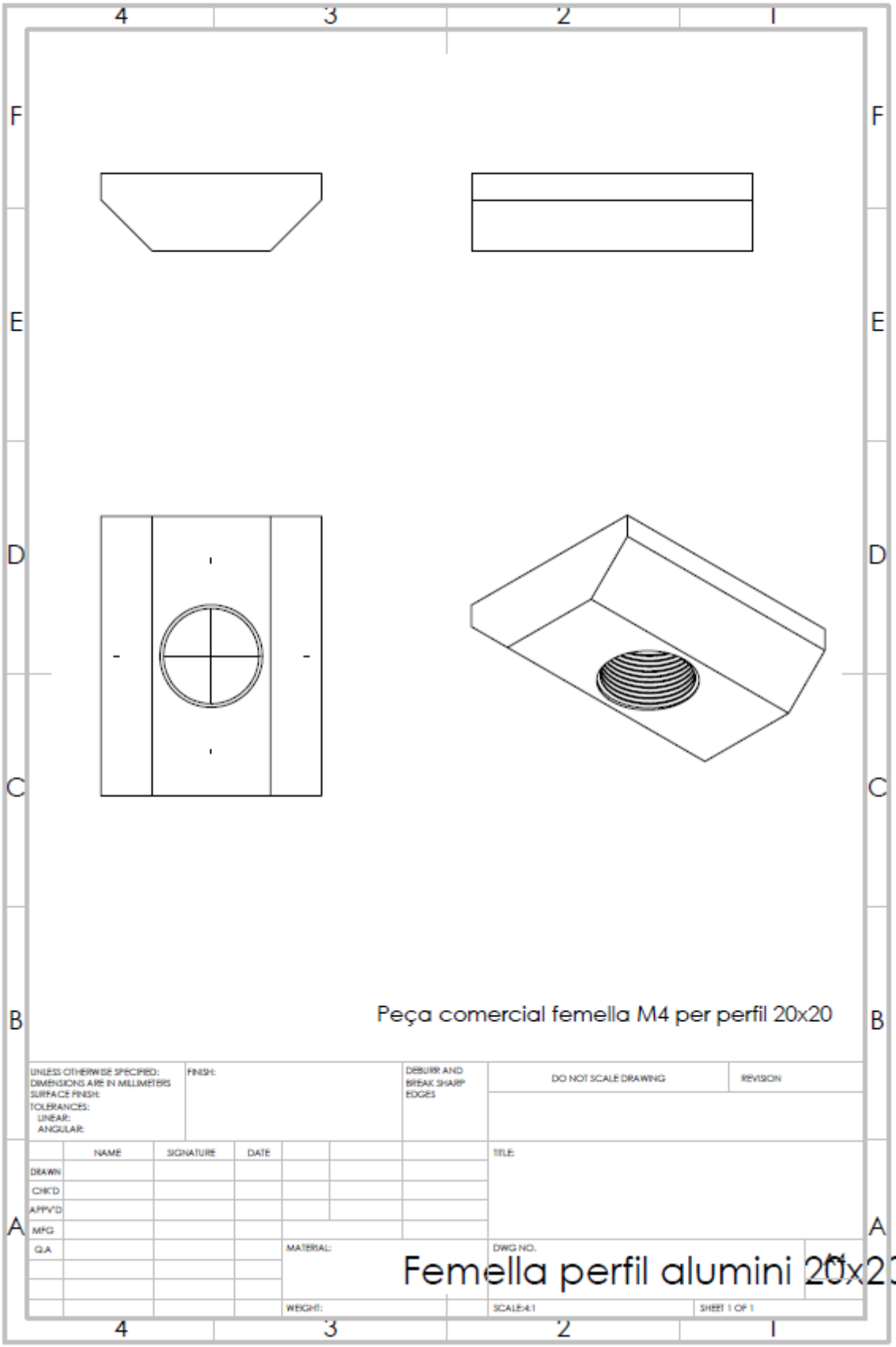


Figura Apèndix A. 18 *Plànol femella perfil alumini*

A.19 Plànol escaire perfil alumini

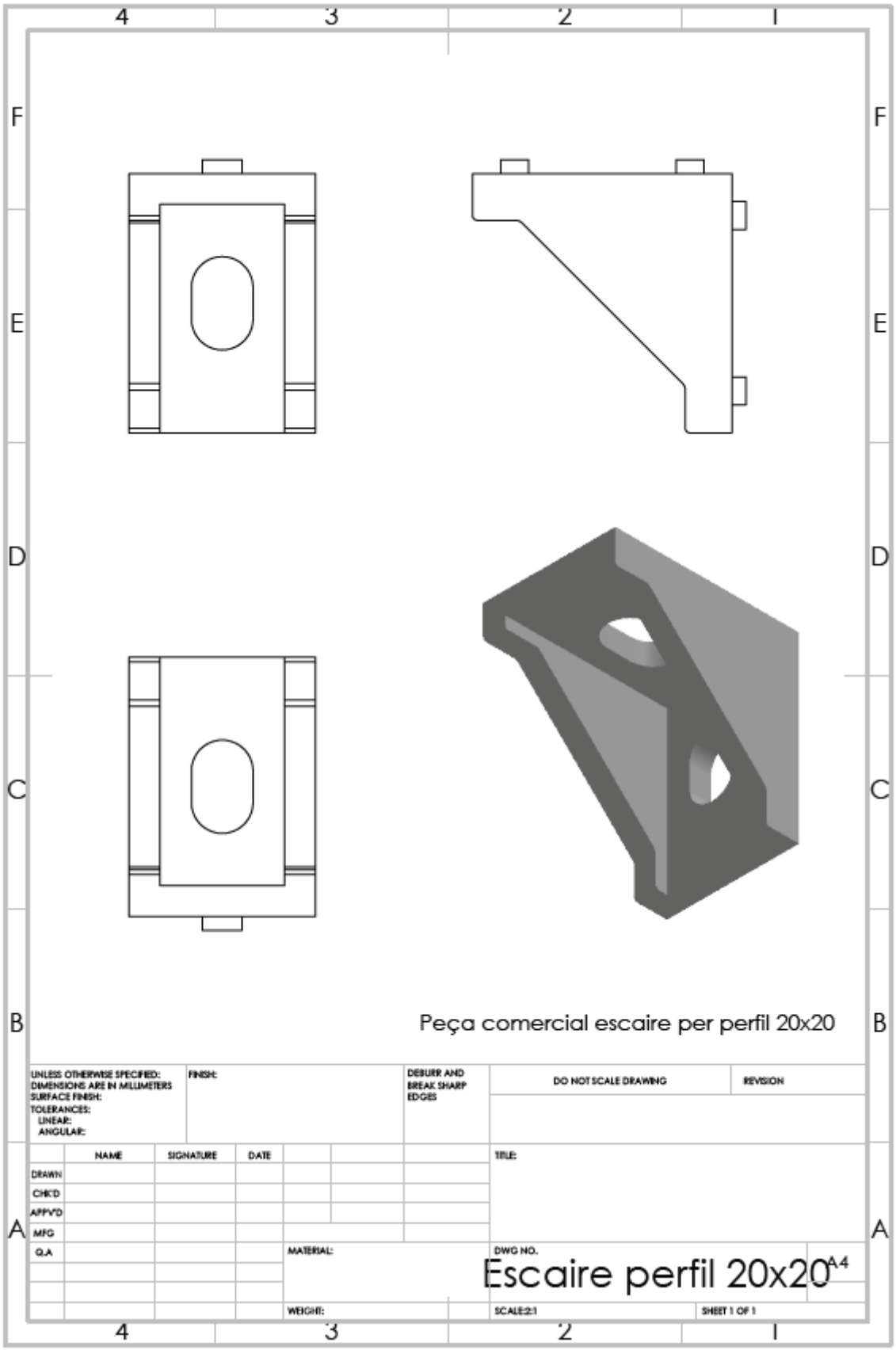


Figura Apèndix A. 19 *Plànol escaire perfil alumini 20x20*

A.20 Plànol Arduino Due

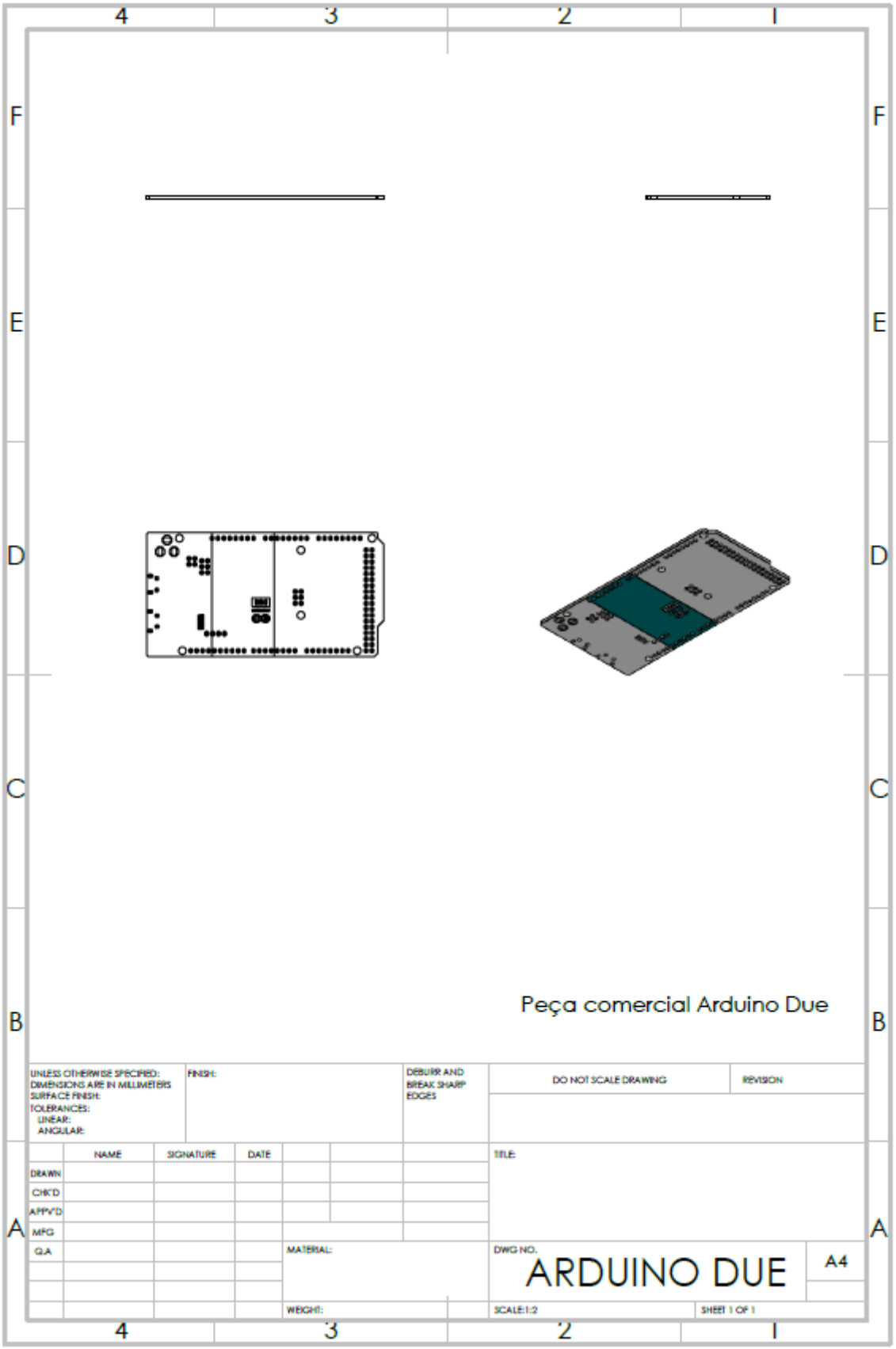


Figura Apèndix A. 20 *Plànol Arduino Due*

A.21 Plànol Servomotor

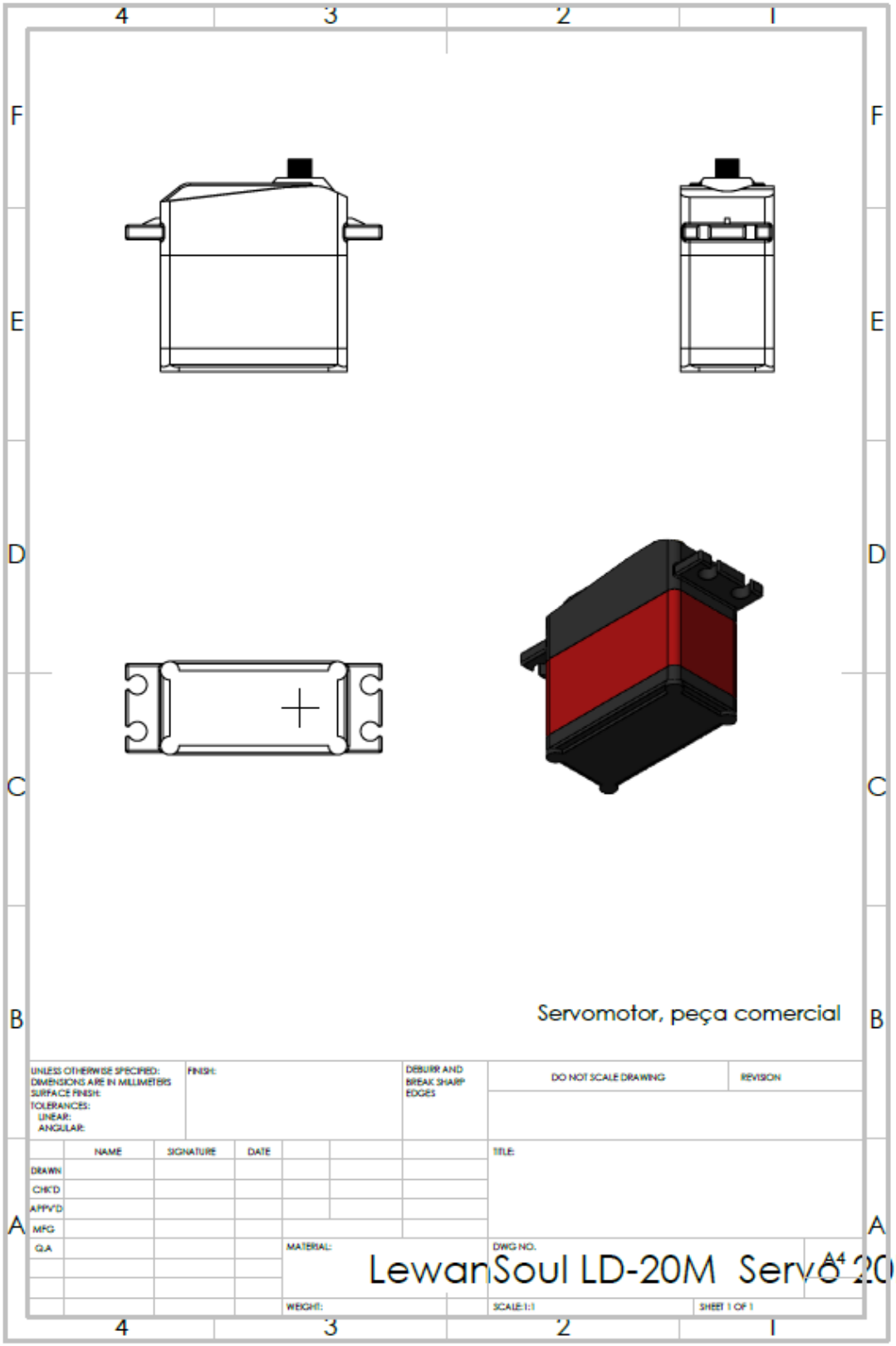


Figura Apèndix A. 21 *Plànol Servomotor*

Apèndix B Codi Arduino IDE

```
//Include Libraries

#include "elapsedMillis.h"
#include <Servo.h>
#include <math.h>

Servo myservo;                  // Providing name the servo motor as servo

//Defining variables
boolean START = false;
boolean i = false;
elapsedMillis mainloop;
float delta = 0, dt ;
float Acc = 0, B = 0, K, M, dx = 0, F, T, xinit = 0.2, initial_angle;
float x = xinit;
float valor_pos_servo, angleservo, angle_servo_escalat;
float vaac;
int admissible_speed;
float degrees_check, inverted_x, degrees_inverted_x, degree_difference,
half_T, needed_speed, max_speed;
String message, M_message, message_send ;
float T_max, max_K, max_M ;
String max_K_str="", max_M_str="";


void setup() {
    Serial.begin(115200);       //Initialization of the serial port with
baud rate of 115200
    while (!Serial);            //Wait until the serial port starts
    myservo.attach(2);          //Attaches the servo on pin 2 to the servo
object
    myservo.write(2);           //Set the servo in the initial position
}

void loop(){
    inici_programa:                 //Here is where the code goes when we
press reset button
    delta = 0;                      //We set delta variable the mainloop
value
    //We set the important variables as 0
    mainloop = 0;
    dt = 0;
    F = 0;
    Acc = 0;
    dx = 0;
    x = 0;
    SerialReceive();                          //Function to read the received
data
    delay(1000);
    check_freq();                   //We check if the mechanism could
simulate with the data entered by the user
    delay(2000);
    i = true;  //We set the i boolean variable as true
    while (i == true) {                         //While boolean variable i is
true
      if(Serial.available()){                 //If there is data to read
        char bytechar = Serial.read();          //We assign the data read in
the bytechar variable
```

```
        if(bytechar == '/'){
          START = true;       //If the bytechar variable == / we set the
boolean variable START as true
          mainloop = 0;       //We put the mainloop time at 0
        }
        if(bytechar == '*'){      //If bytechar variable == * (STOP button)
          i = false;              //We put i and START as false
          START = false;
          goto inici_programa;    //We go to the start of the loop
          }
      }

      if (START == true){                     //If START is true we could
start the movement simulation
        if (mainloop > 10) {                  //If mainloop is greater
than 10 we continue with the movement simulation
          delta = mainloop;                   //We set delta variable the
mainloop value
          mainloop = 0;                       //We reset the mainloop
value
          dt = ((float)delta)/ 1000.0;        //Convert the units to I.S.
units and we force delta value to be a float

          Acc = (-K * x - B * dx) / M;        //We compute the
acceleration
          dx += Acc * dt;                     //We integrate the
acceleration the get the speed
          x += dx * dt;                       //We integrate the speed to
get the position

          T = 2 * PI * sqrt(M / K);           //We compute the period
          F = 1 / T;                          //We compute the frequency

          //We send information using serial communication
          Serial.write('*');                  //We send to the serial
port a * to indicate that we have data
          Serial.println(dt);                 //We send the Sample time
          Serial.println(x);                  //We send the
position
          Serial.println(F);                  //We send the frequency

          valor_pos_servo = angle_calculator();  //We assign the value
returned by the angle_calculator function to valor_pos_servo
          myservo.write(int(valor_pos_servo));   //We order the servo to go
to the calculated angle
        }
      }
  }
}

void SerialReceive()                    //Function to read the data
received
  {
  while (!Serial.available());          //Wait until no data is received
    M = Serial.parseFloat();            //Assign to M the first valid
floating point number from the Serial buffer
    xinit = Serial.parseFloat();        //Assign to xinit the first valid
floating point number from the Serial
    B = Serial.parseFloat();            //Assign to B the first valid
floating point number from the Serial buffer
    K = Serial.parseFloat();
```

```
    x = float(xinit*1.000);              //We assign the xinit value to x
and we assign the value as float
  }

float check_freq(){                      //Function that allows to check if
the mechanism could simulate correctly with the
                                           //parameters entered by the user.

  T = 2 * PI * sqrt(M / K);              //We compute the period

  if ((x==0.20) or (x==-0.20)){          //if the x is 0.2 or -0.2, the
degree difference is going to be 180°
    degree_difference = 180;
  }
  else{
    degrees_check = degrees_calculator();                          //We
compute the angle of the initial position
    inverted_x = x * (-1.000);                                     //We
compute which is the inverted x to compute the angle in that position
    degrees_inverted_x = degrees_calculator_inverted_pos();        //We
compute the angle in the inverted x position
    degree_difference = abs(degrees_check - degrees_inverted_x);   //We
compute the absolute value of the differences between the angles, to check
how many degrees has to move the servo.
  }
  half_T = T / 2;                                  //We compute the time to
half period
  needed_speed = degree_difference / half_T;       //We compute the needed
speed to accomplish the period time
  max_speed = 400;                                 //We assign de max_speed
as 400 provided by the manufacturer

  if (needed_speed > max_speed){   //If the needed speed is larger than the
maximum, we set admissible speed as 0
    admissible_speed = 0;
  }
  else{
    admissible_speed = 1;          //Else, we set admissible speed as 1
  }
  if (admissible_speed == 0){                                  //If admissible
speed=0, we have to recalculate the Mass or K
    message="Freq. inadmisible prova: ";
    T_max = 2 * degree_difference / max_speed;             //This is the
maximum period that could spend

    max_M = ((K * pow(T_max,2)) / (4 * pow(PI,2)));        //We compute the
maximum Mass
    max_K = ((M * 4 * pow(3.1415,2)) / (pow(T_max,2)));    //We compute the
maximum K

    max_M_str="";                                          //We initialize
the string as "" (empty)
    max_K_str="";                                          //We initialize
the string as "" (empty)

    max_M_str.concat(max_M);                               //We concatenate
the string
    max_K_str.concat(max_K);                               //We concatenate
the string
```

```
    M_message="";                                                    //W
e initialize the M_message as a "" (empty)
    String M_message = String("Mass>" + max_M_str + "K<" +
max_K_str);  //We create the message

    message_send="";                        //We initialize the message_send
as a "" (empty)
    message_send = message + M_message;  //We concatenate the message
    Serial.println(message_send);        //We send a message with the
maximum Mass and K

    //We set the messages as "" (empty) for the next time
    max_M_str="";
    max_K_str="";
    M_message="";
    message="";
    message_send="";
  }

  if(admissible_speed == 1){                    //If admissible speed == 1,
we send a message that the frequency is correct.
    Serial.println("Frequencia admisible");
  }
    valor_pos_servo = angle_calculator();     //We assign the value
returned by the angle_calculator function to valor_pos_servo
    myservo.write(int(valor_pos_servo));      //We order the servo to go to
the calculated angle
}

float degrees_calculator(){   //Function to compute the angle using the
cosine theorem
  return(acos((-
pow(0.40000,2)+pow(0.20000,2)+pow(x+0.20000+0.20000,2))/(2*0.20000*(x+0.200
00+0.20000)))*57.2957795);   //We return the angle value
}

float degrees_calculator_inverted_pos(){    //Function to compute the angle
usig the cosine theorem
  return(acos((-
pow(0.40000,2)+pow(0.20000,2)+pow(inverted_x+0.20000+0.20000,2))/(2*0.20000
*(inverted_x+0.20000+0.20000)))*57.2957795);   //We return the angle value
}

float angle_calculator() {         //Function to compute the angle
according to the mechanism dimensions and current position

  vaac = (-
pow(0.40000,2)+pow(0.20000,2)+pow(x+0.20000+0.20000,2))/(2*0.20000*(x+0.200
00+0.20000)); //We compute the angle usig the cosine theorem
  angleservo=acos(vaac);                                        //We compute
the arccosinus
  angle_servo_escalat = angleservo * 57.2957795 * 255 / 352; //We adjust
the value for a 270° servo motor

  if (angle_servo_escalat > 179){                                  //If the
value is greater than 179 we assign the value to 180
                                                                     //this
is if the value is 180.2 for example due to decimal
errors
      angle_servo_escalat = 3.1415 * 57.2957795 * 255 / 335;    //We set
the servo value as 180 degrees position
```

```
    }
  return angle_servo_escalat;                                    //We return
the angle in degrees that the servo has to move on
}
```

Apèndix C Codi programació aplicació

```
classdef app_springer_damper_model < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        GridLayout                      matlab.ui.container.GridLayout
        LeftPanel                       matlab.ui.container.Panel
        TotalTimeEditField              matlab.ui.control.EditField
        TotalTimeEditFieldLabel         matlab.ui.control.Label
        TextEditField                   matlab.ui.control.EditField
        FrequencyEditFieldEditLabel     matlab.ui.control.Label
        FrequencyEditField              matlab.ui.control.EditField
        Image3                          matlab.ui.control.Image
        NmLabel                         matlab.ui.control.Label
        KEditField                      matlab.ui.control.EditField
        KEditFieldLabel                 matlab.ui.control.Label
        kgLabel                         matlab.ui.control.Label
        NmsLabel                        matlab.ui.control.Label
        Image2                          matlab.ui.control.Image
        Image                           matlab.ui.control.Image
        SimulationParametersLabel       matlab.ui.control.Label
        Pos_Ini_ValueEditField          matlab.ui.control.EditField
        Pos_IniSlider                   matlab.ui.control.Slider
        Pos_IniSliderLabel              matlab.ui.control.Label
        BEditField                      matlab.ui.control.EditField
        BEditFieldLabel                 matlab.ui.control.Label
        MassEditField                   matlab.ui.control.EditField
        MassEditFieldLabel              matlab.ui.control.Label
        SendDataButton                  matlab.ui.control.Button
        STOPButton                      matlab.ui.control.Button
        STARTButton                     matlab.ui.control.Button
        RightPanel                      matlab.ui.container.Panel
        UIAxes                          matlab.ui.control.UIAxes
    end

    % Properties that correspond to apps with auto-reflow
    properties (Access = private)
        onePanelWidth = 576;
    end


    properties (Access = public)
        springer_damper_model=serialport('COM5', 115200, Timeout=10);    %We
configure and initialize the serial communication

        %Defining variable as a public properties
        DataSample;
        DataPosition;
        DataFrequency;
        Test_Time=0;
        samples=0;
        Sample_time=0;
        Boto=0;
        time;
    end
```

```
    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: STARTButton
        function STARTButtonPushed(app, event)
            %Function that starts when the START button is pushed
            %We initialize the arrays
            app.DataSample=zeros(5,1);
            app.DataPosition=zeros(5,1);
            app.DataFrequency=zeros(5,1);
            app.time=zeros(5,1);
            %We initialize some variables
            app.Test_Time=0;
            app.samples=0;
            app.Sample_time=0;

            write(app.springer_damper_model,'/','char');  % We send a "/"
to the serial port

            %We clear some variables
            clear DataSample;
            clear DataPosition;
            clear DataFrequency;
            clear app.UIAxes;
            clear samples;
            app.samples=0;
            app.Test_Time=0;

             while(app.Boto==1)
                if(app.springer_damper_model.BytesAvailableFcnCount()>0)
                                                %If there is Bytes
available
                    if(read(app.springer_damper_model,1,'char') == '*')
                                                %We read the data and if
it's a "*" we start with the movement
                        app.samples=app.samples+1;
                                                %We increase the number of
samples counter

app.DataSample(app.samples)=str2double(readline(app.springer_damper_model))
;     %We read and convert to a double the sample time data sent from the
arduino code
                        app.Sample_time=10;
                                                            %We set the
sample time as 10us

app.DataPosition(app.samples)=str2double(readline(app.springer_damper_model
));   %We read from the serial port the position data, then whe conveWe
read from the serial port the position, then we convert the data to double
and we save itrt the data to double and we save it

app.DataFrequency(app.samples)=str2double(readline(app.springer_damper_mode
l));  %We read from the serial port the frequency data, then we convert the
data to double and we save it
                        app.Test_Time=app.Test_Time+app.Sample_time;
                %We compute the current test time

app.FrequencyEditField.Value=num2str(app.DataFrequency(app.samples));
%We show the frequency read by the serial port as a string in the frequency
field
```

```
app.TotalTimeEditField.Value=num2str(app.Test_Time/1000);
            %We show the total time read by the serial port as a string in
the total time field
                        app.time(app.samples)=(app.Test_Time/1000);
                  %We save each loop the test time in seconds

plot(app.UIAxes,app.time,app.DataPosition,"LineStyle","-");
%We plot the position Vs the test time
                    end
                end
             end
            write(app.springer_damper_model,'-','char');
                                                    %When the number of
samples is equal to the threshold value we send a - to indicate the final
of the simulation

        end

        % Button pushed function: SendDataButton
        function SendDataButtonPushed(app, event)
            %This function starts when the Send Data button is pushed.
            %When it is pushed we send the mass, the initial position and
            %the dumper constant through the serial port.

            app.Boto=1;
                  %Send data
            writeline(app.springer_damper_model,(app.MassEditField.Value));

writeline(app.springer_damper_model,(num2str(app.Pos_IniSlider.Value)));
            writeline(app.springer_damper_model,(app.BEditField.Value));
            writeline(app.springer_damper_model,(app.KEditField.Value));
            if(app.springer_damper_model.BytesAvailableFcnCount()>0)
            %If there is Bytes available
               app.TextEditField.Value=readline(app.springer_damper_model);
      %We write in the text field the value that we read
            end
        end

        % Button pushed function: STOPButton
        function STOPButtonPushed(app, event)
            %This function starts when the Stop button is pushed.
            %When it is pushed we send a "*" through the serial port to
indicate
            % to the arduino code that the simulation has to stop.
            write(app.springer_damper_model,'*','char');
        end

        % Value changing function: Pos_IniSlider
        function Pos_IniSliderValueChanging(app, event)
            %This function starts when the slider value is changing.
            changingValue = event.Value;                              %We
save the current value
            v=round((app.Pos_IniSlider.Value),3);           %We round the
value with 3 decimals
            app.Pos_Ini_ValueEditField.Value=num2str(v);    %We show this
number in the Pos_Ini field.
        end

        % Changes arrangement of the app based on UIFigure width
        function updateAppLayout(app, event)
```

```
            currentFigureWidth = app.UIFigure.Position(3);
            if(currentFigureWidth <= app.onePanelWidth)
                % Change to a 2x1 grid
                app.GridLayout.RowHeight = {480, 480};
                app.GridLayout.ColumnWidth = {'1x'};
                app.RightPanel.Layout.Row = 2;
                app.RightPanel.Layout.Column = 1;
            else
                % Change to a 1x2 grid
                app.GridLayout.RowHeight = {'1x'};
                app.GridLayout.ColumnWidth = {250, '1x'};
                app.RightPanel.Layout.Row = 1;
                app.RightPanel.Layout.Column = 2;
            end
        end
    end

    % Component initialization
    methods (Access = private)
        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.AutoResizeChildren = 'off';
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.SizeChangedFcn = createCallbackFcn(app,
@updateAppLayout, true);

            % Create GridLayout
            app.GridLayout = uigridlayout(app.UIFigure);
            app.GridLayout.ColumnWidth = {250, '1x'};
            app.GridLayout.RowHeight = {'1x'};
            app.GridLayout.ColumnSpacing = 0;
            app.GridLayout.RowSpacing = 0;
            app.GridLayout.Padding = [0 0 0 0];
            app.GridLayout.Scrollable = 'on';

            % Create LeftPanel
            app.LeftPanel = uipanel(app.GridLayout);
            app.LeftPanel.ForegroundColor = [0.502 0.502 0.502];
            app.LeftPanel.BackgroundColor = [0.502 0.502 0.502];
            app.LeftPanel.Layout.Row = 1;
            app.LeftPanel.Layout.Column = 1;

            % Create STARTButton
            app.STARTButton = uibutton(app.LeftPanel, 'push');
            app.STARTButton.ButtonPushedFcn = createCallbackFcn(app,
@STARTButtonPushed, true);
            app.STARTButton.BackgroundColor = [0.3922 0.8314 0.0745];
            app.STARTButton.FontWeight = 'bold';
            app.STARTButton.Position = [12 133 100 22];
            app.STARTButton.Text = 'START';

            % Create STOPButton
            app.STOPButton = uibutton(app.LeftPanel, 'push');
            app.STOPButton.ButtonPushedFcn = createCallbackFcn(app,
@STOPButtonPushed, true);
            app.STOPButton.BackgroundColor = [1 0 0];
            app.STOPButton.FontWeight = 'bold';
```

```
            app.STOPButton.Position = [131 133 100 22];
            app.STOPButton.Text = 'STOP';

            % Create SendDataButton
            app.SendDataButton = uibutton(app.LeftPanel, 'push');
            app.SendDataButton.ButtonPushedFcn = createCallbackFcn(app,
@SendDataButtonPushed, true);
            app.SendDataButton.BackgroundColor = [1 1 0];
            app.SendDataButton.FontWeight = 'bold';
            app.SendDataButton.Position = [76 174 100 22];
            app.SendDataButton.Text = 'Send Data';

            % Create MassEditFieldLabel
            app.MassEditFieldLabel = uilabel(app.LeftPanel);
            app.MassEditFieldLabel.HorizontalAlignment = 'right';
            app.MassEditFieldLabel.FontWeight = 'bold';
            app.MassEditFieldLabel.Position = [60 399 35 22];
            app.MassEditFieldLabel.Text = 'Mass';

            % Create MassEditField
            app.MassEditField = uieditfield(app.LeftPanel, 'text');
            app.MassEditField.BackgroundColor = [0.8 0.8 0.8];
            app.MassEditField.Position = [110 399 86 22];

            % Create BEditFieldLabel
            app.BEditFieldLabel = uilabel(app.LeftPanel);
            app.BEditFieldLabel.BackgroundColor = [0.502 0.502 0.502];
            app.BEditFieldLabel.HorizontalAlignment = 'center';
            app.BEditFieldLabel.FontWeight = 'bold';
            app.BEditFieldLabel.FontColor = [0.149 0.149 0.149];
            app.BEditFieldLabel.Position = [70 354 25 22];
            app.BEditFieldLabel.Text = 'B';

            % Create BEditField
            app.BEditField = uieditfield(app.LeftPanel, 'text');
            app.BEditField.HorizontalAlignment = 'center';
            app.BEditField.BackgroundColor = [0.8 0.8 0.8];
            app.BEditField.Position = [110 354 85 22];

            % Create Pos_IniSliderLabel
            app.Pos_IniSliderLabel = uilabel(app.LeftPanel);
            app.Pos_IniSliderLabel.HorizontalAlignment = 'right';
            app.Pos_IniSliderLabel.FontWeight = 'bold';
            app.Pos_IniSliderLabel.Position = [12 244 48 22];
            app.Pos_IniSliderLabel.Text = 'Pos_Ini';

            % Create Pos_IniSlider
            app.Pos_IniSlider = uislider(app.LeftPanel);
            app.Pos_IniSlider.Limits = [-0.196 0.196];
            app.Pos_IniSlider.ValueChangingFcn = createCallbackFcn(app,
@Pos_IniSliderValueChanging, true);
            app.Pos_IniSlider.FontWeight = 'bold';
            app.Pos_IniSlider.Position = [82 244 144 3];

            % Create Pos_Ini_ValueEditField
            app.Pos_Ini_ValueEditField = uieditfield(app.LeftPanel,
'text');
            app.Pos_Ini_ValueEditField.BackgroundColor = [0.8 0.8 0.8];
            app.Pos_Ini_ValueEditField.Position = [8 209 56 36];

            % Create SimulationParametersLabel
```

```
app.SimulationParametersLabel = uilabel(app.LeftPanel);
app.SimulationParametersLabel.FontSize = 20;
app.SimulationParametersLabel.FontWeight = 'bold';
app.SimulationParametersLabel.Position = [23 442 224 25];
app.SimulationParametersLabel.Text = 'Simulation Parameters';

% Create Image
app.Image = uiimage(app.LeftPanel);
app.Image.Position = [23 395 34 30];
app.Image.ImageSource = 'Massa.jpg';

% Create Image2
app.Image2 = uiimage(app.LeftPanel);
app.Image2.Position = [20 343 39 33];
app.Image2.ImageSource = 'Dumper-PhotoRoom.png';

% Create NmsLabel
app.NmsLabel = uilabel(app.LeftPanel);
app.NmsLabel.Position = [202 348 45 22];
app.NmsLabel.Text = 'N/(m/s)';

% Create kgLabel
app.kgLabel = uilabel(app.LeftPanel);
app.kgLabel.Position = [205 399 25 22];
app.kgLabel.Text = 'kg';

% Create KEditFieldLabel
app.KEditFieldLabel = uilabel(app.LeftPanel);
app.KEditFieldLabel.BackgroundColor = [0.502 0.502 0.502];
app.KEditFieldLabel.HorizontalAlignment = 'center';
app.KEditFieldLabel.FontWeight = 'bold';
app.KEditFieldLabel.FontColor = [0.149 0.149 0.149];
app.KEditFieldLabel.Position = [70 299 25 22];
app.KEditFieldLabel.Text = 'K';

% Create KEditField
app.KEditField = uieditfield(app.LeftPanel, 'text');
app.KEditField.HorizontalAlignment = 'center';
app.KEditField.BackgroundColor = [0.8 0.8 0.8];
app.KEditField.Position = [110 299 85 22];

% Create NmLabel
app.NmLabel = uilabel(app.LeftPanel);
app.NmLabel.Position = [208 299 27 22];
app.NmLabel.Text = 'N/m';

% Create Image3
app.Image3 = uiimage(app.LeftPanel);
app.Image3.Position = [11 295 50 30];
app.Image3.ImageSource = 'Spring.jpg';

% Create FrequencyEditField
app.FrequencyEditField = uieditfield(app.LeftPanel, 'text');
app.FrequencyEditField.BackgroundColor = [0.8 0.8 0.8];
app.FrequencyEditField.Position = [86 96 100 22];

% Create FrequencyEditFieldEditLabel
app.FrequencyEditFieldEditLabel = uilabel(app.LeftPanel);
app.FrequencyEditFieldEditLabel.HorizontalAlignment = 'right';
app.FrequencyEditFieldEditLabel.FontWeight = 'bold';
app.FrequencyEditFieldEditLabel.Position = [10 96 66 22];
```

```
            app.FrequencyEditFieldEditLabel.Text = 'Frequency';

            % Create TextEditField
            app.TextEditField = uieditfield(app.LeftPanel, 'text');
            app.TextEditField.HorizontalAlignment = 'center';
            app.TextEditField.FontSize = 10;
            app.TextEditField.BackgroundColor = [0.8 0.8 0.8];
            app.TextEditField.Position = [8 6 236 47];

            % Create TotalTimeEditFieldLabel
            app.TotalTimeEditFieldLabel = uilabel(app.LeftPanel);
            app.TotalTimeEditFieldLabel.HorizontalAlignment = 'right';
            app.TotalTimeEditFieldLabel.FontWeight = 'bold';
            app.TotalTimeEditFieldLabel.Position = [9 63 64 22];
            app.TotalTimeEditFieldLabel.Text = 'Total Time';

            % Create TotalTimeEditField
            app.TotalTimeEditField = uieditfield(app.LeftPanel, 'text');
            app.TotalTimeEditField.BackgroundColor = [0.8 0.8 0.8];
            app.TotalTimeEditField.Position = [88 63 100 22];

            % Create RightPanel
            app.RightPanel = uipanel(app.GridLayout);
            app.RightPanel.ForegroundColor = [0.502 0.502 0.502];
            app.RightPanel.BackgroundColor = [0.502 0.502 0.502];
            app.RightPanel.Layout.Row = 1;
            app.RightPanel.Layout.Column = 2;

            % Create UIAxes
            app.UIAxes = uiaxes(app.RightPanel);
            title(app.UIAxes, 'Position Vs Time')
            xlabel(app.UIAxes, 'X')
            ylabel(app.UIAxes, 'Y')
            zlabel(app.UIAxes, 'Z')
            app.UIAxes.FontWeight = 'bold';
            app.UIAxes.FontSize = 14;
            app.UIAxes.Position = [1 6 383 473];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end
    % App creation and deletion
    methods (Access = public)
        % Construct app
        function app = app_springer_damper_model
            % Create UIFigure and components
            createComponents(app)
            % Register the app with App Designer
            registerApp(app, app.UIFigure)
            if nargout == 0
                clear app
            end
        end
        % Code that executes before app deletion
        function delete(app)
            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

Apèndix D Esquema de connexions

En aquest apartat es mostra l'esquema de connexions entre la placa controladora Ardunio Due i el servomotor. Es pot apreciar a la Figura Apèndix D. 1, que la comunicació entre ambdós dispositius s'estableix mitjançant el pin D2PWM, en el qual es trameten les ordres de control del servomotor. Per a l'alimentació del servomotor, aquest es troba connectat a una font d'alimentació externa de 6.8V compartint negatiu comú amb l'Arduino Due.

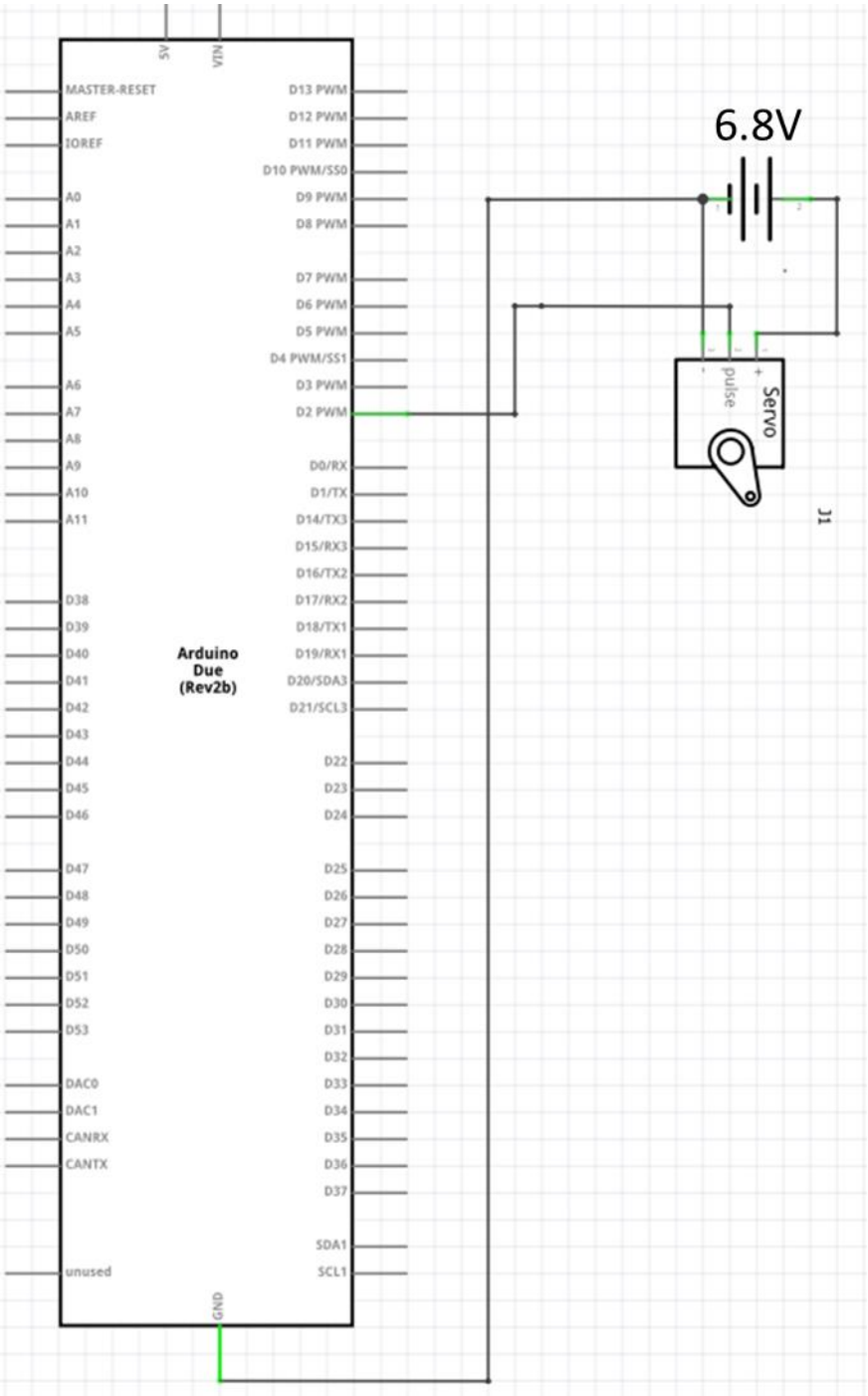


Figura Apèndix D. 1 *Esquema de connexions*